Ann Oper Res (2012) 196:53–71
DOI 10.1007/s10479-012-1084-7

A tree search algorithm for solving the multi-dimensional strip packing problem with guillotine cutting constraint

Andreas Bortfeldt · Sabine Jungmann

Published online: 23 February 2012
© Springer Science+Business Media, LLC 2012

Abstract The article presents a tree search algorithm (TRSA) for the strip packing problem in two and three dimensions with guillotine cutting constraint. In the 3D-SPP a set of rectangular items (boxes) and a container with fixed width and height but variable length are given. An arrangement of all boxes within the container has to be determined so that the required length is minimised. The 2D-SPP is analogously defined. The proposed TRSA is based on a tree search algorithm for the container loading problem by Fanslau and Bortfeldt (INFORMS J. Comput. 22:222–235, 2010). The TRSA generates guillotine packing patterns throughout. In a comparison with all recently proposed 3D-SPP methods the TRSA performs very competitive. Fine results are also achieved for the 2D-SPP.

Keywords Strip packing · Open dimension problem · Guillotine cutting · Tree search

1 Introduction

This article deals with the rectangular strip packing problem (SPP) in two and three dimensions, respectively. In the 3D-SPP a set of rectangular items (boxes) and a container with fixed width and height but variable length are given. A feasible arrangement of all boxes within the container has to be determined so that the required length is minimised. The 2D-SPP is analogously defined. Note that the 2D-SPP can be perceived as a 3D-SPP in which all boxes have a minimum dimension of one length unit while the container height also equals one.

An arrangement is called feasible if no items overlap and all boxes are placed completely within the container and parallel to the container walls. There are no further restrictions imposed for the item stock, so the stock may be homogeneous (one single type of boxes), weakly heterogeneous (a few box types, many boxes per type) or strongly heterogeneous (many box types, a few boxes per type). The items of a box type represent congruent parallelepipeds.

A. Bortfeldt (✉) · S. Jungmann
Department of Information Systems, University in Hagen, 58084 Hagen, Germany
e-mail: andreas.bortfeldt@fernuni-hagen.de

According to the typology of cutting and packing problems (C&P) by Wäscher et al. (2007) the SPP considered here is referred to as open dimension problem (ODP), or more precisely as 3D (or 2D) rectangular open dimension problem with one variable dimension.

The 3D-SPP and the 2D-SPP occur in different industries and in many practical cutting and packing scenarios. In the steel industry, one common problem is to cut small three-dimensional blocks that belong to one customer order out of a larger steel block. Often the profile of the large block is fixed, while its length is variable. Customers usually order identical sets of pieces repeatedly. In such a situation, a good planning of the cutting process is important in order to limit the length of the large block and so to minimize the trim loss. Similar 2D cutting problems can be found in the glass- and paper industry, in which rectangles of known dimensions are to be cut out of a strip with a given width, but variable length.

The following packing problem is in fact also an SPP. Modern attached freight systems aim at a better capacity utilization of trucks. Such a system collects orders of different senders for trucks of different carriers. An order can only be assigned to a truck of sufficient residual capacity determined by the residual free length. Thus, before the assignment, the system calculates the minimum additional length required by a given consignment. Similar algorithms can improve the design of containers, or can even help the selection of vehicles in a vehicle fleet for transporting a given volume of goods.

In practical cutting and packing applications a wide variety of constraints can be observed (cf. Bischoff and Ratcliff 1995). Here, the following three constraints are considered:

(C1) *Orientation constraint*: For a defined subset of the item stock the permitted orientation variants are restricted. For each box up to five of the original six possible spatial orientation variants can be ruled out in the 3D case while in the 2D case one of the two possible orientations may be prohibited.

(C2) *Guillotine cutting constraint*: It requires that all items of a packing arrangement can be reproduced by guillotine cuts, i.e. by wall-to-wall cuts that run parallel to the walls of the container.

(C3) *Support constraint*: The bottom area of each packed box that is not placed on the container floor must be supported completely (i.e. 100%) by other boxes.

The first two constraints play an important role in two-dimensional and three-dimensional C&P applications (cf., e.g., Lodi et al. 1999). In particular the guillotine cutting constraint is often enforced in cutting processes by the available equipment (saws etc.). The support constraint is an important stability condition in 3D (but not in 2D) packing applications to prevent boxes from tilting.

In this article we propose a tree search algorithm (TRSA) for solving the 3D-SPP and the 2D-SPP. The TRSA results by a straightforward adaptation of a tree search algorithm for the 3D container loading problem (CLP) that was proposed by Fanslau and Bortfeldt (2010). Note that in the 3D-CLP (considered here) a box set and one container with fixed dimensions are given and the aim is to maximize the volume (or value) of the packed boxes while generally not all given boxes can be packed. The TRSA generates guillotine patterns throughout, i.e. constraint (C2) is always observed. Both of the other constraints introduced above are satisfied if necessary. The main issue to be investigated is whether the high performance of the integrated CLP method is transferred to the SPP method. The TRSA will be subjected to a comparison with other recently published SPP solution procedures and it will turn out that the TRSA is able to generate top-quality solutions in reasonable running times particularly in the 3D case.

The rest of the article is organized as follows. First, in Sect. 2 we give a literature overview concerning the SPP in three and two dimensions. In Sect. 3 the SPP algorithm is described including an outline of the underlying CLP algorithm as well as its adaptation to the strip packing problem. In Sect. 4 we compare the proposed SPP method to other algorithms from literature by numerical experiments while some conclusions are drawn in Sect. 5.

2 Literature review

The SPP is NP-hard (cf. Hopper and Turton 2001). Exact algorithms for the 2D-SPP were proposed by Martello et al. (2003), Lesh et al. (2004), Bekrar et al. (2007), Bekrar and Kacem (2009) and Kenmochi et al. (2009). A general framework for exactly solving multi-dimensional packing problems was developed by Fekete and Schepers (1997) (see also Fekete et al. 2007). At the present time only smaller SPP instances can be solved exactly; typical sizes of exactly solved 2D instances range up to 50 items. Hence, heuristic approaches (that do not guarantee optimal solutions) are indispensable for solving large SPP instances in the 2D and even more in the 3D case.

For the 2D-SPP a wide variety of heuristic solution methods has been proposed in recent years. The interested reader is referred to Hopper and Turton (2000, 2001) and to Riff et al. (2009) for a comprehensive survey. We can distinguish four 2D-SPP subtypes (variants) regarding the constraints (C1) and (C2) that are tackled by 2D-SPP algorithms (cf. Lodi et al. 1999):

- OF: orientation of all pieces is fixed (O) and guillotine cutting is not required (F);
- RF: pieces may be rotated by 90° (R) and guillotine cutting is not required (F);
- OG: orientation of all pieces is fixed (O) and guillotine cutting is required (G);
- RG: pieces may be rotated by 90° (R) and guillotine cutting is required (G).

Note that a feasible solution with regard to subtype OG is also feasible regarding the other subtypes while a feasible solution with regard to subtype RG and OF, respectively, is also feasible in terms of subtype RF. In Table 1 a representative sample of heuristic 2D-SPP algorithms is presented. The sample is focused on the 2D-SPP with guillotine cutting constraint but recent and most relevant papers regarding subtypes OF and RF are also considered. For each contribution author(s) and year, relevant subtype(s), type of solution approach and additional features of the algorithm are listed.

The subtypes without guillotine cutting constraint (OF, RF) draw much more attention than the other subtypes (OG, RG). This becomes even clearer if the above mentioned survey articles are consulted. Most of the algorithms apply metaheuristic search strategies like genetic algorithms or simulated annealing. At the moment, the solution methods proposed by Alvarez-Valdes et al. (2008), Belov et al. (2008) and Burke et al. (2009) belong to the most powerful approaches for the 2D-SPP without guillotine cutting constraint (OF, RF) while the genetic algorithm by Bortfeldt (2006) and the branch and bound method by Cui et al. (2008) achieved the best results so far for the guillotine cutting subtypes OG and RG. It is noticeable that three of these five algorithms are based on widely used metaheuristic strategies (cf. Table 1). On the other hand, it turned out that straightforward constructive heuristics are of great value in the design of high-performance solution methods and often successful 2D-SPP methods are built around those placement heuristics (see Table 1, rightmost column). Clearly, the well-known Bottom Left (BL) heuristic (cf. Baker et al. 1980) can be regarded as the prototype of such placement heuristics.

In contrast to the 2D-SPP, only few algorithms have been developed for solving the 3D-SPP so far and almost all of them are listed in Table 2. Table 2 is similarly built as Table 1

Table 1 Heuristic algorithms for the 2D-SPP (sample)

Authors, source	2D-SPP subtype	Type of approach	Additional characteristics
Kröger (1993, 1995)	RG	parallel GA	encoding by binary trees, problem specific operators
Schnecke (1996)	RG	parallel GA	similar to Kröger (1995)
Mumford-Valenzuela et al. (2004)	RG	GA	encoding by normalized postfix representation
Burke et al. (2004)	RF	CH	Best Fit heuristic
Lesh et al. (2005)	OF	LS	possible enhancement by human interaction
Bortfeldt (2006)	all	GA	no encoding, problem specific operators
Zhang et al. (2006)	RG	LS	recursive procedure
Zhang et al. (2005)	RG	SA	built around recursive placement heuristic, see Zhang et al. (2006)
Cui et al. (2008)	RG	B&B	recursive procedure
Alvarez-Valdes et al. (2008)	OF	GRASP	reactive GRASP, parameter learning mechanism
Belov et al. (2008)	OF	LS	uses “sequential value correction” (SVC) approach and 1D heuristics SubKP and Bottom-Left-Right (BLR)
Burke et al. (2009)	RF	SA	includes heuristics Best Fit (BF) and Bottom Left Fill (BLF)
Wei et al. (2009)	RF	RLS	includes heuristic Least Wasted First (LWF)
Ortmann et al. (2010)	all	CH	new and improved constructive heuristics
Allen et al. (2011)	RF	TS	includes heuristics Best Fit (BF) and Deepest Bottom Left Fill (DBLF)

GA: genetic algorithm; CH: constructive heuristic, SA: simulated annealing, TS: tabu search, GRASP: greedy randomized adaptive search procedure, B&B: branch and bound, (R)LS: (random) local search

but column 2 indicates now whether the methods are able to observe the support constraint (C3).

Again, some metaheuristic methods include straightforward constructive heuristics. On the other hand, algorithms for the 3D-SPP are often based on existing algorithms for the 3D-CLP (e.g., see Table 2, branch and bound method of Bortfeldt and Mack 2007).

3 The strip packing algorithm

In the following, the tree search algorithm, referred to as SPTRS (Strip Packing by Tree Search), is introduced in two steps. At first the frame procedure of SPTRS is explained, afterwards the integrated CLP method is described.

3.1 The overall algorithm

The overall algorithm of method SPTRS is shown in Fig. 1. The main idea is to solve a given SPP instance by solving a series of CLP instances by means of a CLP algorithm. The

Table 2 Heuristic algorithms for the 3D-SPP (for acronyms see Table 1)

Authors, source	Support constraint (C3) observed?	Type of approach	Additional characteristics
Bischoff and Mariott (1990)	No	CH	includes multiple alternative greedy heuristics, container is filled by consecutive "walls"
Sixt (1996)	No	TS	uses order based encoding and straightforward placement heuristic
Bortfeldt and Gehring (1999)	Yes	parallel GA, parallel TS	methods result by modification of CLP methods
Karabulut and Inceoglu (2004)	No	GA	includes heuristic Deepest Bottom Left Fill (DBLF)
Bortfeldt and Mack (2007)	No	B&B	built around CLP method by Pisinger (2002)
Allen et al. (2011)	No	TS	includes heuristics 3D Best Fit (3BF) and Deepest Bottom Left Fill (DBLF)

Fig. 1 Overall algorithm of method SPTRS

```
input (problem data, parameters)
set container length lC := lClB * 100 / minspp_fillrate
solve CLP instance
if no SPP solution found, i.e. if not all boxes were packed then
    lC := high_value
    solve CLP instance
endif
repeat
    lC := lCUsed – 1 // lCUsed of previous SPP solution
    solve CLP instance
until no SPP solution found
output (last found SPP solution).
```

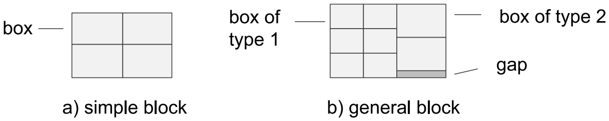
box set, the container width wC and the container height hC of all CLP instances are given by the SPP instance. Starting with a sufficiently large value (see below) the container length lC is reduced from instance to instance. The solving of CLP instances will only be finished after no SPP solution has been returned for the first time. A solution of a CLP instance is called an SPP solution if and only if all given boxes were packed.

Each time an SPP solution for a CLP instance with container length lC has been calculated, the so-called used container length $lCUsed$ ($lCUsed \le lC$), given by the maximum coordinate of a box in length direction, is determined. In the next CLP instance the container length is then set $lC := lCUsed - 1$. In this way it is ensured that a new SPP solution is always better than the previous one. Finally, the last (i.e. best) found SPP solution is output.

The initially chosen container length should meet two different demands. On the one hand, it has to be chosen so that an SPP solution (including all given boxes) can be generated in any case. On the other hand, the initial container length should be not too large in order to save computational effort that is caused by a long series of solved CLP instances. Both the demands are satisfied as follows.

First, the initial container length is calculated as product of the continuous (or material) lower bound $lClB$ of the required container length and a factor greater than one. The lower bound is determined as $lClB := \lceil(\text{total box volume})/(wC^{*}hC)\rceil$ ($\lceil a\rceil$ is the smallest integer not smaller than a). The factor is given as quotient $100/minspp\_fillrate$ wherein the param-

Fig. 2 A simple block and a general block (top view, 2D case)



eter *minspp_fillrate* stands for the expected minimum filling rate of the container volume (in %). With a suitable value of *minspp_fillrate*, e.g. 90%, a very large initial container length can be avoided. Hence, an attempt is made to solve the related CLP instance.

Only if this attempt fails, a second greater initial container length *lC* is tried. This time *lC* is set to the sum of the minimum dimensions in length direction of all boxes (this sum is denoted as *high_value* in Fig. 1). The minimum dimension of a box in length direction is determined considering the container dimensions as well as the orientation constraint (C1). Clearly, the CLP with length *high_value* has an SPP solution in any case, formed by a sequence of all boxes in length direction, so that the first requirement is also met. Hence, an initial SPP solution with a related length *lCUsed* is available in any case.

Note that the computation is (also) finished if the used container length *lCUsed* of the last found SPP solution reaches the lower bound *lClB* (not shown in Fig. 1).

3.2 The integrated CLP algorithm

To solve the CLP instances the tree search algorithm CLTRS (Container Loading by Tree Search) by Fanslau and Bortfeldt (2010) is used. CLTRS generates guillotine packing patterns throughout, i.e. constraint (C2) is always satisfied, while orientation and support constraint (C1) and (C3), respectively, are optionally observed. Below we give an extended outline of algorithm CLTRS; for a comprehensive description we refer the reader to Fanslau and Bortfeldt (2010).

Algorithm CLTRS is based on the so-called block building approach. Instead of filling the container by single boxes it is packed by blocks of several boxes without or only with small gaps. Compared to older block building algorithms, CLTRS does not only use loss-free blocks consisting of boxes of the same box type which are arranged in the same spatial orientation (so-called simple blocks). Instead the block concept is extended in that also general blocks are allowed that comprise boxes of different types or have congruent boxes with different spatial orientations. A general block may have gaps (cf. Fig. 2). However, the minimum volume utilization of a block (regarding the enveloping cuboid) is stipulated by a parameter that is usually set to a high value, e.g. 98%. General blocks are mainly beneficial for strongly heterogeneous problem instances.

The overall algorithm of CLTRS is shown in Fig. 3. The search is organized in two stages. In the first stage only simple blocks are used while in the second stage general blocks are allowed. At the beginning of each stage an appropriate list of oriented blocks, that serve to fill the container, is generated in advance. Each stage has a specific time limit. The best packing plan over both stages is output at the end. Using the two stages can be considered a means to diversify the search.

In each stage a series of complete solutions (packing plans) is generated. The admitted search effort per solution is controlled by an internal parameter *search_effort* that is doubled from solution to solution. Hence, better and better solutions can be found within a search stage. A solution is constructed block by block and each block fills a residual space, i.e. an empty block-shaped space of defined size and position inside the container. The residual spaces are collected in a stack that is initially filled by the container itself, i.e. the interior space of the container is the first residual space to be processed. To process the current

```
input (problem data, parameters)
for stage := 1 to 2 do // carry out search in two stages
    generate special block list for current stage
    initialize search effort per solution: search_effort := 1
    repeat // generate successive packing plans with increasing search effort
        initialize search state (packing plan, unpacked boxes, stack of residual spaces)
        while stack of residual spaces not empty do
            determine best block for uppermost residual space
            update search state
        endwhile
        update search effort for next solution: search_effort := search_effort * 2
    until time_limit(stage) is exceeded
endfor
output (best packing plan over both stages).
```

Fig. 3 Overall algorithm of method CLTRS

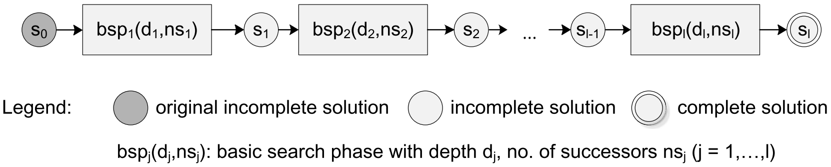

s0
bsp1(d1,ns1)
s1
bsp2(d2,ns2)
s2
...
sl-1
bspl(dl,nsl)
sl
Legend:
original incomplete solution
incomplete solution
complete solution
bspj(dj,nsj): basic search phase with depth dj, no. of successors nsj (j = 1,…,l)

Fig. 4 A partition search and its basic search phases

residual space, namely the uppermost residual space in the stack, the most appropriate block is selected from the block list and then placed in the residual space's lower corner nearest to the origin of the used coordinate system. Afterwards the rest of the space is orthogonally cut resulting in up to three new residual spaces that are again inserted into the stack. Heuristic rules for cutting (filled) and inserting (new) residual spaces aim at preventing waste, i.e. no longer usable space in the container. If a block has been specified and placed, this decision is not revised when further blocks are determined.

Determining the best block for the current residual space forms the kernel operation of a search stage and is based on the following principle. Let an incomplete solution s_0 including n_0 ($n_0 \geq 0$) placed blocks be given. To find a block for the uppermost residual space *rstop* of the residual space stack, starting from s_0, i.e. retaining the n_0 blocks already placed, different complete solutions are generated experimentally. The $(n_0 + 1)$th block of all temporarily-generated solutions fills the (same) residual space *rstop* that results by the previously added blocks. The $(n_0 + 1)$th block of the complete temporary solution with maximal packed volume is then returned as the best block for *rstop*.

Each of the complete solutions results by a separate tree search (called *partition search*) that is itself composed by several basic search phases (cf. Fig. 4). The basic search phases are chained as the incomplete solution s_j that is output by the jth basic search phase is taken as input solution by the $(j + 1)$th phase. Of course, the input solution of the first basic search phase is the original incomplete solution s_0. Each basic search phase adds d_j ($d_j \geq 1$) blocks to its input solution ($j = 1, \ldots, l$). Only the last basic search phase provides a complete solution.

In the jth basic search phase a tree is constructed in which the input solution s_{j-1} and the related state data (residual space stack, set of unpacked boxes) form the root node. The tree is defined by two parameters, namely the depth d_j ($d_j \geq 1$) and the number of successor nodes ns_j. Starting with the root, ns_j successor nodes are constructed per node (search

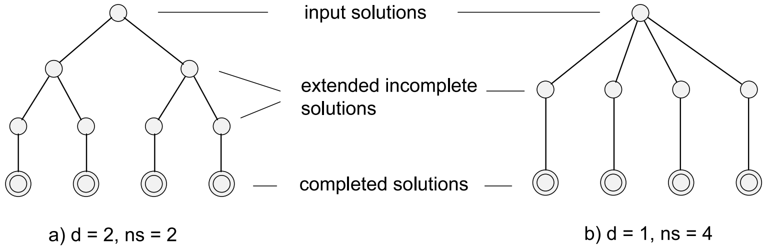


Fig. 5 Two different search trees of a basic search phase

state) by filling the respective uppermost residual space alternately with the largest blocks (in terms of volume) that fit into the space and consist of still unpacked boxes. Since d_j blocks are to be added the tree will have a depth of $d_j + 1$ and generally each extended solution will have $n_{j-1} + d_j$ blocks if the input solution consists of n_{j-1} blocks.

The extended solutions are generally still incomplete and to determine the best extended solution of the jth basic search phase all extended solutions are completed by the so-called greedy completion. Given an incomplete solution s' the greedy completion packs repeatedly the largest possible block into the respective uppermost residual space until no further block can be packed, i.e. a complete solution s'' is reached. Now the output solution of the jth basic search phase is determined as the extended solution s' of (generally) $n_{j-1} + d_j$ blocks that yields the best complete solution s'' with maximum packed volume. Only in the last basic search phase the best completed solution s'' is returned instead of the related incomplete solution s' (see Fig. 4).

The number of successor nodes *ns* in a basic search phase is determined as a function of the depth d so that a greater depth d corresponds to a smaller successor number *ns* and vice versa. In Fig. 5 two possible trees of a basic search phase are shown. While in the first tree of depth $d = 2$ the number of successors per node is $ns = 2$, the second tree has the parameters depth $d = 1$ and $ns = 4$. Note that the number of successors *ns* determines the search width or local diversity of the search while the depth d affects the number of simultaneously considered consecutive search levels, i.e. the degree of foresight within a basic search phase.

If the best block for a residual space has to be determined, at first the total depth *td* of the partition searches is determined as a function of the parameter *search_effort*, i.e. *td* is increased with *search_effort*. Then all ordered (number-theoretic) partitions of integer *td, denoted by* $\pi = (l, d_1, d_2, \ldots, d_l)$ with $l \geq 1$, $d_j > 0$, $j = 1, \ldots, l$ and $\sum_{j=1}^{l} d_j = td$, are considered. For each such partition π a partition search is carried out with l basic search phases and depth values d_j given by π. As explained above the best complete solution over all partition searches then determines the block for the current residual space. If, e.g., the total depth is $td = 3$, then partition searches have to be done for the four partitions $(1, 3)$, $(2, 1, 2)$, $(2, 2, 1)$ and $(3, 1, 1, 1)$.

The special fashion of tree search outlined before is called partition controlled tree search (PCTRS). According to the experience with the CLP the PCTRS is able to handle two success factors of a tree search, namely the search width and the degree of foresight, in a flexible and balanced way.

To adapt the CLP algorithm to the SPP the main routine is modified (see Fig. 3). If an SPP solution including all given items was found in the first stage then the search for the given container length is aborted, i.e. the second stage is ignored. It has been experimentally proved that in this way the search effort can be reduced without worsening the solution quality.

4 Computational experiments

SPTRS was implemented in C/C++ using the GNU C/C++ compiler together with Netbeans IDE 6.8 as development environment. All tests were run on an Intel 3.16 GHz PC with 3.6 GB RAM (Intel Core2 Duo, CPU 8500 3.16 GHz). In the computational experiments the following parameter settings were used throughout. The parameter *minspp_fillrate* was set to 90% (cf. Sect. 3.1). For the parameters of method CLTRS the original settings as described in Fanslau and Bortfeldt (2010) were adopted. However, a pair of reduced time limits for the search stages (10 seconds for stage 1, 20 seconds for stage 2) was applied throughout. As explained above the second stage is only performed if no SPP solution was found in the first stage of the search for a given container length. Moreover, in the 3D case a time limit of 80 seconds for the entire search is imposed (commented in detail below). Afterwards the experimental results for the 3D and 2D cases are presented.

4.1 Results on 3D test instances

To evaluate the performance of SPTRS for the 3D-SPP, almost all problem instances introduced by Bortfeldt and Gehring (1999) and by Bortfeldt and Mack (2007) were calculated. Altogether, five different instance sets for the 3D-SPP with a total of 800 instances are used (note that a test case is a subset of an instance set):

- Instance set SP-BR was introduced in Bortfeldt and Gehring (1999) (see also Bortfeldt and Mack 2007) and includes 10 test cases of 10 instances each. The instances were derived from the 3D-CLP instances proposed by Bischoff and Ratcliff (1995) and Davies and Bischoff (1998), respectively. The number of box types per instance varies from 3 in test case 1 to 50 in test case 10. The mean number of boxes per instance and test case lies between 128 (in test case 5) and 140 (in test case 2). The character of the test cases changes progressively from weakly to strongly heterogeneous. The optimal container lengths are not known.
- Instance set SP-BR-XL was introduced in Bortfeldt and Mack (2007) and includes 10 test cases of 10 instances each. The problem instances and test cases are similarly constructed as for instance set SP-BR. However the number of boxes per instance amounts constantly to 1000 and the instances can be viewed as weakly heterogeneous throughout. The optimal container lengths are not known.
- Instance sets L, G and N were introduced in Bortfeldt and Mack (2007). Each of the sets includes 8 test cases with 25 instances each. For all three instance sets the number of box types per instance varies from 8 in test case 1 to 50 in test case 8 of the instance set. The mean number of boxes per instance lies between 120 and 200 boxes. Each instance of instance set L is constructed in such a way that an optimal solution with a volume utilization of 100% (perfect solution) made up of some walls that follow each other in length direction of the container is known. Each instance of set G has a known perfect solution, observing guillotine cutting constraint (C2) that has not a wall structure in general. Finally, each instance of instance set N has a known perfect solution not observing (C2).

The results for the 3D-SPP instances are shown in Tables 3, 4, 5 that are explained as follows:

Table 3 Results for instance set SP-BR (best utilizations for case (C3) not required in bold)

Method	TSA-CC-4P	SPBBL-CC4	3BF	3BF+TS	SPTRS	SPTRS	SPTRS	SPTRS
100% support	Yes	No	No	No	No	No	Yes	Yes
Utilization type	*VU* (%)	*VU* (%)	VU_2 (%)	VU_2 (%)	*VU* (%)	VU_2	*VU*	VU_2
SP-BR01 (3)	92.3	87.3	88.7	90.0	**94.0**	**94.1**	93.6	93.7
SP-BR02 (5)	93.5	88.6	89.0	89.6	**94.5**	**94.6**	94.5	94.5
SP-BR03 (8)	92.3	89.4	87.8	89.0	**94.4**	**94.5**	93.9	94.0
SP-BR04 (10)	90.8	90.1	87.8	88.8	**93.7**	**93.8**	92.6	92.7
SP-BR05 (12)	89.9	89.3	87.7	88.5	**93.3**	**93.3**	92.0	92.1
SP-BR06 (15)	89.2	89.7	87.6	88.6	**93.2**	**93.3**	91.5	91.5
SP-BR07 (20)	87.1	89.2	87.4	88.7	**92.1**	**92.2**	90.3	90.4
SP-BR08 (30)	84.0	87.9	86.8	88.3	**91.0**	**91.0**	87.9	88.0
SP-BR09 (40)	80.9	87.3	86.5	87.9	**90.7**	**90.8**	83.1	83.2
SP-BR10 (50)	79.1	87.6	86.3	87.9	**90.2**	**90.3**	81.1	81.2
Mean utilization	87.9	88.7	87.6	88.7	**92.7**	**92.8**	90.1	90.1
Mean calc. time	234 s	99 s	1 s	160 s	78 s	–	74 s	–

- For instance sets SP-BR and SP-BR-XL the numbers of box types per test case are indicated in brackets in column 1 of the related table. For instance sets L, G and N these numbers are part of the names of the test cases shown in column 1 of Table 5.
- Two SPTRS variants were tested, i.e. SPTRS was applied with and without support constraint (C3). Whether (C3) is observed by SPTRS or by another solution method is indicated in row "100% support" ("yes" or "no").
- Four algorithms from literature were included for a comparison: the parallel tabu search method TSA-CC-4P by Bortfeldt and Gehring (1999), the branch and bound method SPBBL-CC4 by Bortfeldt and Mack (2007), the constructive heuristic 3BF and the tabu search method 3BF+TS by Allen et al. (2011). These can be viewed as the most effective methods so far (cf. Table 2). The following computers were used for the compared methods: TSA-CC-4P: four Pentium PCs (200 MHz each), SPBBL-CC4: AMD Athlon (2.0 GHz), 3BF and 3BF+TS: Intel PC (1.86 GHz).

  A time limit for the *entire* search of SPTRS was imposed to ensure that the computational effort used for SPTRS shows only small differences to the effort spent by the recent competing methods SPBBL-CC4 and 3BF+TS, respectively. Considering the specified time limit of 160 seconds for both the methods 3BF+TS and SPBBL-CC4 and the given data for the processors used, the time limit for SPTRS was set to 80 seconds. Here it was taken into account that this time limit of SPTRS is often slightly exceeded since the elapsed time is only checked outside the integrated container loading method. Mean computing times of SPTRS and all compared methods can be found in Tables 3–10.
- The solution quality is given as volume utilization (*VU*) of the container in percent that is calculated as $VU = (\text{total box volume})/(wC^*hC^*lCUsed)$ (in %). However, Allen et al. (2011) calculate the volume utilization as $VU_2 = lClB/lCUsed$ (in %) (cf. Sect. 3.1 for calculation of $lClB$). It is easily seen that $VU_2 \geq VU$ in any case (since $\lceil a \rceil \geq a$). Both utilizations, *VU* and VU_2, are indicated for the SPTRS variants in Tables 3 and 4. In Table 5 only utilization *VU* is indicated since the existence of perfect solutions ensures the identicalness of VU_2 and VU for all instances of the sets L, G and N.
- As SPTRS is a deterministic algorithm, the SPTRS results are compared to *mean* volume utilizations of stochastic algorithms if mean and best utilization values are available. Per

Table 4 Results for instance set SP-BR-XL (best utilizations for case (C3) not required in bold)

Method	SPBBL-CC4	3BF	3BF+TS	SPTRS	SPTRS	SPTRS	SPTRS
100% support	No	No	No	No	No	Yes	Yes
Utilization type	*VU* (%)	VU_2 (%)	VU_2 (%)	*VU* (%)	VU_2	*VU*	VU_2
SP-BR01-XL (3)	86.9	92.2	92.4	**95.9**	**95.9**	95.9	95.9
SP-BR02-XL (5)	88.3	92.2	92.4	**95.6**	**95.6**	95.6	95.6
SP-BR03-XL (8)	89.8	91.8	91.9	**94.9**	**94.9**	94.3	94.3
SP-BR04-XL (10)	90.2	92.0	92.1	**94.5**	**94.5**	94.2	94.3
SP-BR05-XL (12)	89.9	92.4	92.5	**94.1**	**94.1**	93.6	93.6
SP-BR06-XL (15)	91.5	92.5	92.6	**93.5**	**93.5**	92.9	92.9
SP-BR07-XL (20)	91.0	92.4	92.6	**92.8**	**92.9**	92.1	92.1
SP-BR08-XL (30)	90.8	92.6	**92.8**	92.5	92.5	80.4	80.4
SP-BR09-XL (40)	90.9	92.1	**92.3**	92.2	92.2	90.7	90.8
SP-BR10-XL (50)	90.4	92.5	**92.7**	92.3	92.3	90.5	90.5
Mean utilization	90.0	92.3	92.4	**93.8**	**93.9**	92.0	92.0
Mean calc. time	157 s	20 s	160 s	93 s	–	90 s	–

method and test case or instance set the mean volume utilization over all instances is presented.

- Calculation times are given in seconds (related to used processor(s)) and represent *total* computation times (not times to best-values). Calculation times are averaged over all instances of a test case or instance set. Alternatively time limits are used as calculation times.

The results for the 3D-SPP can be summarized as follows:

- Generally, the SPTRS variant that fulfills the support constraint (C3) achieves smaller volume utilizations than the other variant. For instance set SP-BR the observance of (C3) "costs" approximately 2.6 %-points and for instance set SP-BR-XL approximately 1.8%-points of volume utilization. These results correspond to the findings of Fanslau and Bortfeldt (2010) on the influence of constraint (C3). However, for nine groups of instances with smaller numbers of box types within the instance sets L, G and N an anomaly regarding this influence could be observed (see Table 5).

  A particular high solution quality can be achieved by SPTRS for weakly heterogeneous problem instances while the solution quality decreases as the heterogeneity of the box sets increases. Note that this is not the case with, e.g., method SPBBL-CC4 (cf., e.g., Table 3).
- To ensure a fair comparison in first instance only methods that behave in the same way regarding support constraint (C3) should be compared to each other. Hence, the SPTRS variants that observe (C3) should be compared to the parallel tabu search method TSA-CC-4P while the SPTRS variants that do not observe (C3) should be compared to the other methods taken from literature. If constraint (C3) is required SPTRS reaches a better solution quality than the competing method TSA-CC-4P for all 10 test cases of the instance set SP-BR (for other instance sets no results are available for competing methods observing (C3)). If constraint (C3) is not required SPTRS achieves a better or equal solution quality than all competing methods for 28 of 44 test cases. In 3 test cases of instance set SP-BR-XL the competing method 3BF+TS performs better on average and in 13 test cases of the instance sets L, N and G, respectively, the method SPBBL-CC4

Table 5 Results for instance sets L, G and N (best utilizations for case (C3) not required in bold)

Method	SPBBL-CC4	SPTRS	SPTRS
100% support	No	No	Yes
	VU (%)	*VU* (%)	*VU* (%)
L08	96.4	**99.8**	99.7
L10	97.3	**98.4**	98.7
L12	97.3	**98.1**	97.7
L16	95.8	**96.7**	96.6
L20	**97.7**	96.7	96.0
L30	**97.9**	95.9	95.1
L40	**97.2**	95.1	94.3
L50	**96.4**	95.1	94.6
L, mean utilization	**97.0**	**97.0**	96.6
L, mean calc. time	160 s	63 s	60 s
G08	94.5	**96.0**	97.9
G10	94.1	**95.9**	96.6
G12	94.8	**95.1**	95.8
G15	**94.4**	93.9	94.1
G20	**94.6**	93.4	93.1
G30	**94.5**	92.4	91.7
G40	**95.5**	91.7	90.8
G50	**95.1**	90.9	86.2
G, mean utilization	**94.7**	93.7	93.3
G, mean calc. time	160 s	83 s	80 s
N08	93.5	**95.9**	97.1
N10	94.5	**95.1**	95.7
N12	94.4	**94.6**	94.7
N15	**93.8**	**93.8**	94.2
N20	**93.4**	93.1	92.8
N30	**93.4**	91.5	90.8
N40	**92.9**	90.6	88.5
N50	**93.1**	90.6	83.2
N, mean utilization	**93.6**	93.1	92.1
N, mean calc. time	160 s	82 s	82 s

achieves the better average utilization. Best results of competing methods are failed for some test cases with higher numbers of box types (see Tables 4 and 5). Averaged over all 200 instances of the sets SP-BR and SP-BR-XL SPTRS achieves a mean volume utilization of 93.3% while the mean utilizations of 3BF+TS and SPBBL-CC4 amount to 90.6% and 89.3%, respectively. The mean volume utilization over all 800 instances of all considered instance sets is 94.3% for SPTRS and amounts to 93.7% for SPBBL-CC4. All in all, SPTRS proves to be very competitive in the 3D case. The mean total calculation times in Tables 3–5 show that the computational effort of SPTRS and the best competing methods is similar.

4.2 Results on 2D test instances

The performance of SPTRS for the 2D-SPP is tested using four instance sets with 616 instances:

- Instance set SP-KR was introduced by Kröger (1993, 1995) and includes 12 strongly heterogeneous instances with 25 to 60 items. The number of item types corresponds nearly to the number of items and optimal container lengths are not known.
- Instance set SP-HT was defined by Hopper (2000) and by Hopper and Turton (2000) and comprises three series C, N and T with 21, 35 and 35 strongly heterogeneous instances, respectively. Each series consists of 7 test cases of 3 or 5 instances with (nearly) the same number of items. All instances have perfect optimal solutions without any waste (at least if SPP subtype RF is supposed), i.e. the lower bound *lClB* (see Sect. 3.1) specifies the optimal container length. The instances of series T have guillotineable optimal solutions while the instances of series N have only non-guillotineable optimal solutions.
- Instance set SP-N was generated by Burke et al. (2004) and consists of 13 strongly heterogeneous instances with 10 to 3152 rectangles. Each instance has a perfect solution so that the optimal container lengths are known.
- Instance set SP-BWMV comprises ten classes (test cases) with 50 instances each. The first six classes (C01 to C06) were suggested by Berkey and Wang (1987) and the last four classes (C07 to C10) were defined by Martello and Vigo (1998) (see Iori et al. 2002, for a fine description). Each class has five subclasses with 10 instances each. The item numbers per instance lie between 20 and 100 and all instances are strongly heterogeneous. Optimal container lengths are not known.

Algorithm SPTRS has been tested in two variants, namely for the SPP subtypes OG and RG. In the first variant the orientation of all items is fixed (as stipulated in all tested 2D instances) while in the second variant rotating the items is allowed. In the interest of a fair comparison CLTRS is primarily matched to 2D-SPP methods from literature that also observe the guillotine cutting constraint, i.e. that are specified for SPP subtypes OG or RG. Only if for a given set of instances results of such methods are not available, top-quality procedures for subtypes OF or RF are considered. As in the 3D case SPTRS is compared preferably to mean (not best) results of stochastic methods and total calculation times are given in seconds for the used processor. Unlike the 3D case no time limit for the entire search of method SPTRS is specified for the calculation of 2D instances since there are too many competing methods that were tested using very different testbeds.

The results for instance set SP-KR can be found in Table 6. For each instance the number of items is given in brackets (column 1). SPTRS is compared to the parallel genetic algorithms by Kröger (1993, 1995) and Schnecke (1996), respectively. Further competing methods are the genetic algorithm SPGAL by Bortfeldt (2006) and the B&B method HRBB by Cui et al. (2008). SPGAL has been tested on a Pentium PC with 2 GHz while HRBB results were achieved on a PC with 2.8 GHz. All competing methods work for subtype RG. In Table 6 the used container lengths (*lCUsed*) are presented. SPTRS (RG) performs somewhat better than the parallel GAs and HRBB and it performs slightly worse than SPGAL. While SPTRS achieves a better quality for the smaller instances SPGAL dominates for the larger ones.

Table 7 shows the results for instance set SP-HT, series C. For each test case (of three instances) the number of items per instance is given in brackets (column 1). SPTRS is compared to the recursive procedure HR and the simulated annealing method SA+HR by Zhang et al. (2005, 2006), respectively, the genetic algorithm SPGAL by Bortfeldt (2006) and to

Table 6 Results (used container lengths) for instance set SP-KR (best results for type RG in bold)

Method	Parallel GA (Schnecke)	Parallel GA (Kröger)	SPGAL	HRBB	SPTRS	SPTRS
SPP subtype	RG	RG	RG	RG	OG	RG
SP-KR01 (25)	111.8	109.4	110.9	111	113	**109**
SP-KR02 (25)	107.6	105.0	105.2	106	107	**105**
SP-KR03 (25)	106.8	104.2	105.2	107	106	**104**
SP-KR04 (35)	158.3	153.0	153.0	154	155	**153**
SP-KR05 (35)	127.5	**123.4**	124.0	125	125	124
SP-KR06 (35)	128.4	**124.6**	125.4	127	128	125
SP-KR07 (45)	202.8	197.0	196.1	199	200	**196**
SP-KR08 (45)	172.8	165.2	**164.9**	166	168	165
SP-KR09 (45)	139.2	135.2	**135.0**	137	137	**135**
SP-KR10 (60)	259.6	253.8	**251.5**	253	257	254
SP-KR11 (60)	288.0	280.8	**277.2**	280	282	278
SP-KR12 (60)	294.6	284.6	**281.9**	284	289	284
Mean lCUsed	174.8	169.7	**169.2**	170.7	172.3	169.3
Mean calc. time	–	–	156 s	66 s	86 s	72 s

Table 7 Results (mean gaps in %) for instance set SP-HT, series C (best results for type RG in bold)

Method	HR	SA+HR	HRBB	SPGAL	SPTRS	SPTRS
SPP subtype	RG	RG	RG	OG	OG	RG
SP-HT-C1 (16-17)	8.3	5.0	**1.7**	3.2	3.3	**1.7**
SP-HT-C2 (25)	4.5	4.5	**0.0**	3.3	2.2	**0.0**
SP-HT-C3 (28-29)	6.7	2.2	1.1	3.9	1.1	**0.0**
SP-HT-C4 (49)	2.2	2.2	2.2	3.8	1.7	**1.7**
SP-HT-C5 (72-73)	1.9	1.9	1.9	2.4	1.5	**1.1**
SP-HT-C6 (97)	2.5	2.5	1.4	1.9	1.4	**0.8**
SP-HT-C7 (196-197)	1.8	3.2	1.3	1.7	1.4	**1.0**
Mean gap	4.0	3.1	1.4	2.9	1.8	**0.9**
Mean calc. time	5 s	345 s	2 s	143 s	58 s	61 s

the B&B method by Cui et al. (2008). HR has been tested on a Dell GX260 PC with 2.4 GHz while HR+SA was run on a DELL GX270 with 3.0 GHz CPU. While SPGAL is tailored to subtype OG the other competing methods work for subtype RG.

In Table 7 the mean gaps are given per test case where the gap of an instance's solution is calculated as $gap = (lCUsed - lClB)/lClB$ (in %) (cf. Sect. 3.1). Both SPTRS variants perform better than the competing methods. SPTRS(RG) improves the mean gap values of HR, SA+HR and HRBB by 3.1, 2.2 and 0.5%-points, respectively, while SPTRS (OG) reaches an improvement of 1.1%-points compared to SPGAL (OG). However, it must be stated that HRBB reaches a good solution quality within very short calculation times compared to the other methods.

Table 8 includes the results for instance set SP-HT, series N and T. For each test case (of five instances) the number of items per instance is given in brackets (columns 1, 6).

Table 8 Results (mean gaps in %) for instance set SP-HT, series N and T

Method	SVC (SubKP)	GRASP	SPTRS	SPTRS	–	SVC (SubKP)	GRASP	SPTRS	SPTRS
SPP subtype	OF	OF	OG	RG	–	OF	OF	OG	RG
N1 (17)	3.3	0.9	10.0	3.5	T1 (17)	0.9	0.0	14.9	4.0
N2 (25)	3.4	3.3	6.2	2.9	T2 (25)	3.5	3.2	6.2	2.9
N3 (29)	3.5	3.6	5.6	2.7	T3 (29)	3.3	3.7	5.3	3.0
N4 (49)	2.5	3.0	3.8	2.0	T4 (49)	2.5	3.0	3.8	2.0
N5 (73)	2.1	2.6	3.4	1.6	T5 (73)	2.1	2.4	3.2	1.7
N6 (97)	1.7	2.2	2.3	1.4	T6 (97)	1.6	2.1	2.2	1.2
N7 (197)	1.0	1.3	1.3	0.7	T7 (199)	1.0	1.5	1.6	1.5
N, mean gap	2.5	2.4	4.7	2.1	T, mean gap	2.1	2.3	5.3	2.3
N, mean calc. time	50 s	60 s	66 s	74 s	T, mean calc. time	50 s	60 s	73 s	69 s

Table 9 Results (volume utilizations *VU* in %) for instance set SP-N (best results for type RF in bold)

Method	BF+SA	3BF+TS	SPTRS	SPTRS
SPP subtype	RF	RF	OG	RG
N1 (10)	**100.0**	**100.0**	**100.0**	**100.0**
N2 (20)	**100.0**	**100.0**	**100.0**	**100.0**
N3 (30)	98.0	**100.0**	98.0	98.0
N4 (40)	97.6	**99.8**	98.8	98.8
N5 (50)	97.1	**99.1**	98.0	98.0
N6 (60)	98.0	**99.3**	99.0	99.0
N7 (70)	96.2	98.0	98.0	**99.0**
N8 (80)	97.6	97.6	**98.8**	**98.8**
N9 (100)	98.7	98.7	98.7	**99.3**
N10 (200)	98.7	**99.3**	**99.3**	**99.3**
N11 (300)	98.0	**99.3**	98.7	98.7
N12 (500)	98.0	**99.0**	98.0	98.0
N13 (3152)	99.6	99.6	**99.9**	**99.9**
SP-N, mean utilization	98.3	98.9	98.9	**99.0**
SP-N, mean calc. time	60 s	60 s	82 s	75 s

SPTRS is compared to method SVC(SubKP) by Belov et al. (2008) and to the GRASP algorithm by Alvarez-Valdes et al. (2008) that are both tailored to subtype OF. SVC(SubKP) was executed on an AMD PC with 2.4 GHz while the results for the GRASP algorithm were calculated on a Pentium 4 Mobile PC with 2 GHz. Again, mean gaps are given per test case. Obviously, the observance of the guillotine cutting constraint (C2) restricts the achievable solutions especially for the smaller instances while this is to some extent compensated by allowing the boxes to be rotated for SPTRS(RG).

Table 9 includes the results for instance set SP-N. For each instance the number of items is indicated in brackets (column 1). SPTRS is compared to method BF+SA by Burke et al. (2009) and 3BF+TS by Allen et al. (2011) that are designed for subtype RF. BF+SA was calculated on a Pentium 4 PC with 2 GHz while Allen et al. (2011) used an Intel PC

Table 10 Results (used container lengths) for instance set SP-BWMV

Method SPP subtype	SVC (SubKP) OF	GRASP OF	SPTRS OG	SPTRS RG
C01-1 (20)	61.4	61.3	62.0	58.4
C01-2 (40)	122.0	121.9	123.2	116.7
C01-3 (60)	188.5	188.7	191.0	180.1
C01-4 (80)	262.6	262.9	269.0	249.7
C01-5 (100)	304.9	305.6	328.0	301.2
C01-Avg	187.9	188.1	194.6	181.2
C02-1 (20)	19.8	19.8	19.8	19.7
C02-2 (40)	39.1	39.1	39.3	39.1
C02-3 (60)	60.1	60.2	60.7	60.1
C02-4 (80)	83.2	83.2	83.6	83.2
C02-5 (100)	100.5	100.5	101.1	100.5
C02-Avg	60.5	60.6	60.9	60.5
C03-1 (20)	164.6	163.5	168.4	157.7
C03-2 (40)	333.9	333.8	339.9	316.8
C03-3 (60)	506.8	506.6	521.8	489.6
C03-4 (80)	709.8	710.0	734.3	679.3
C03-5 (100)	839.5	840.2	882.6	819.9
C03-Avg	510.9	510.8	529.4	492.7
C04-1 (20)	63.9	63.4	65.2	62.7
C04-2 (40)	125.9	126.3	127.8	125.4
C04-3 (60)	195.6	196.7	198.2	195.3
C04-4 (80)	270.4	272.2	274.4	270.7
C04-5 (100)	325.4	327.3	329.7	326.3
C04-Avg	196.2	197.2	199.1	196.1
C05-1 (20)	537.4	533.9	542.8	498.4
C05-2 (40)	1076.5	1074.7	1.086.9	1004.0
C05-3 (60)	1648.1	1645.9	1.675.5	1564.8
C05-4 (80)	2288.5	2290.4	2.372.9	2175.4
C05-5 (100)	2652.1	2652.0	2.821.9	2625.6
C05-Avg	1640.5	1639.4	1700.0	1573.6
C06-1 (20)	168.6	167.3	174.7	164.2
C06-2 (40)	332.2	333.6	336.7	331.1
C06-3 (60)	516.9	520.6	522.7	516.2
C06-4 (80)	714.0	718.9	722.5	717.3
C06-5 (100)	860.5	865.4	870.7	865.6
C06-Avg	518.4	521.2	525.5	518.9
C07-1 (20)	501.9	501.9	504.1	436.8
C07-2 (40)	1059.9	1059.0	1062.9	945.0
C07-3 (60)	1530.0	1529.6	1541.7	1384.0
C07-4 (80)	2222.1	2222.2	2315.5	2031.1
C07-5 (100)	2644.0	2645.2	2772.1	2420.1
C07-Avg	1591.6	1591.6	1639.3	1443.4
C08-1 (20)	461.5	458.0	472.1	448.8

Table 10 (*Continued*)

Method SPP subtype	SVC (SubKP) OF	GRASP OF	SPTRS OG	SPTRS RG
C08-2 (40)	956.1	954.4	980.0	952.3
C08-3 (60)	1400.8	1405.9	1426.6	1400.9
C08-4 (80)	1964.8	1973.6	2009.2	1988.1
C08-5 (100)	2423.1	2439.5	2487.3	2454.2
C08-Avg	1441.3	1446.3	1475.0	1448.9
C09-1 (20)	1106.8	1106.8	1108.4	987.9
C09-2 (40)	2190.6	2190.6	2193.2	1954.4
C09-3 (60)	3410.4	3410.4	3427.8	3041.1
C09-4 (80)	4588.1	4588.1	4621.4	4141.1
C09-5 (100)	5434.9	5434.9	5491.4	5144.4
C09-Avg	3346.2	3346.2	3368.4	3053.8
C10-1 (20)	351.4	350.5	354.3	338.3
C10-2 (40)	667.2	664.5	671.5	647.9
C10-3 (60)	936.0	935.5	947.5	918.3
C10-4 (80)	1211.2	1209.7	1228.0	1196.2
C10-5 (100)	1513.9	1515.1	1544.0	1508.2
C10-Avg	935.9	935.1	949.1	921.8
SP-BWMV mean lCUsed	1042.9	1043.6	1064.1	989.1
SP-BWMV mean calc. time	50 s	60 s	208 s	288 s

with 1.86 GHz. The utilizations are given according to the *VU* formula in Sect. 4.1. Allen et al. (2011) calculate the volume utilization for instance set SP-N as VU_2 (see Sect. 4.1). However, the existence of perfect solutions ensures the identicalness of VU_2 and VU for all SP-N instances. The best mean volume utilization so far has been slightly improved by the SPTRS (RG) variant and the SPTRS (OG) variant is also not dominated by the competing methods that do not satisfy the guillotine cutting and the orientation constraint.

Table 10 presents the results for instance set SP-BWMV. For each class and subclass the constant number of boxes per instance is indicated in column 1. SPTRS is again compared to method SVC(SubKP) by Belov et al. (2008) and to the GRASP algorithm by Alvarez-Valdes et al. (2008) that are tailored to subtype OF. Per subclass or class mean used container lengths are shown. SPTRS (RG) achieves equal or better solutions than the GRASP algorithm for 47 of the 50 subclasses while SPTRS (OG) generally does not reach the solution quality of the compared methods for subtype OF. This result goes with the findings for instance set SP-HT, series N and T.

All in all, SPTRS achieves a top solution quality in reasonable computation times also for 2D-SPP instances. In terms of solution quality SPTRS seems to be at least on a par with the most effective methods for solving the 2D-SPP with guillotine cutting constraint, namely SPGAL and HRBB. A high solution quality was also proved in comparison with high-quality methods for subtype RF using instance set SP-N.

5 Conclusions

In this article an algorithm for the multi-dimensional strip packing problem with guillotine cutting constraint is presented that is based on the method by Fanslau and Bortfeldt (2010) for solving the container loading problem. Solving the SPP is reduced to solving a series of CLP instances with decreasing container lengths. A preferably small value is chosen for the initial container length to shorten the search. The chosen approach is advantageous in some regards. It is easy to implement and the observance of constraints is transferred from the CLP method to the SPP method. The main question to be investigated was whether the high performance of the integrated CLP method is also transferred to the SPP method. This question could be answered in the affirmative by means of extensive numerical experiments. In a comparison to the best known 3D-SPP solution methods the proposed SPP algorithm, called SPTRS, achieved a high solution quality using a similar computational effort as the compared methods. In 28 of 44 3D-SPP test cases SPTRS reached the best volume utilization on average. While SPTRS always satisfies the guillotine cutting constraint this is not the case for most of the competing 3D-SPP methods. Moreover, the experiments proved that algorithm SPTRS belongs to the most effective methods for the 2D-SPP with guillotine cutting constraint.

References

Allen, S. D., Burke, E. K., & Kendall, G. (2011). A hybrid placement strategy for the three-dimensional strip packing problem. *European Journal of Operational Research*, *209*, 219–227.

Alvarez-Valdes, R., Parreno, F., & Tamarit, J. M. (2008). Reactive GRASP for the strip-packing problem. *Computers and Operations Research*, *35*, 1065–1083.

Baker, B. S., Coffmann, E. G., & Rivest, R. L. (1980). Orthogonal packings in two dimensions. *SIAM Journal on Computing*, *9*, 846–855.

Bekrar, A., & Kacem, I. (2009). An exact method for the 2D guillotine strip packing problem. *Advances in Operations Research*, *2009*, 732010.

Bekrar, A., Kacem, I., & Chu, C. (2007). A comparative study of exact algorithms for the two dimensional strip packing problem. *Journal of Industrial and Systems Engineering*, *1*, 151–170.

Belov, G., Scheithauer, G., & Mukhacheva, E. A. (2008). One-dimensional heuristics adapted for two-dimensional rectangular strip packing. *The Journal of the Operational Research Society*, *59*, 823–832.

Berkey, J. O., & Wang, P. Y. (1987). Two dimensional finite bin packing algorithms. *The Journal of the Operational Research Society*, *38*, 423–429.

Bischoff, E. E., & Mariott, M. D. (1990). A comparative evaluation of heuristics for container loading. *European Journal of Operational Research*, *44*, 267–276.

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

Bortfeldt, A. (2006). A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces. *European Journal of Operational Research*, *172*, 814–837.

Bortfeldt, A., & Gehring, H. (1999). Two metaheuristics for strip packing problems. In D. K. Despotis & C. Zopounidis (Eds.), *Proceedings of the fifth international conference of the decision sciences institute*, Athens 1999 (Vol. 2, pp. 1153–1156).

Bortfeldt, A., & Mack, D. (2007). A heuristic for the three-dimensional strip packing problem. *European Journal of Operational Research*, *183*, 1267–1279.

Burke, E. K., Kendall, G., & Whitwell, G. (2004). A new placement heuristic for the orthogonal stock-cutting problem. *INFORMS Journal on Computing*, *52*, 655–671.

Burke, E. K., Kendall, G., & Whitwell, G. (2009). A simulated annealing enhancement of the best-fit heuristic for the orthogonal stock cutting problem. *INFORMS Journal on Computing*, *21*, 505–516.

Cui, Y., Yang, Y., Cheng, X., & Song, P. (2008). A recursive branch-and-bound algorithm for the rectangular guillotine strip packing problem. *Computers & Operations Research*, *35*, 1281–1291.

Davies, A. P., & Bischoff, E. E. (1998). Weight distribution considerations in container loading (Technical Report). European Business Management School, University of Wales, Swansea, Statistics and OR Group.

Fanslau, T., & Bortfeldt, A. (2010). A tree search algorithm for solving the container loading problem. *INFORMS Journal on Computing*, *22*, 222–235.

Fekete, S. P., & Schepers, J. (1997). On more-dimensional packing III: Exact algorithms (Technical Report ZPR97-290). Mathematisches Institut, Universität zu Köln.

Fekete, S. P., Schepers, J., & van der Veen, J. C. (2007). An exact algorithm for higherdimensional orthogonal packing. *Operations Research*, *55*, 569–587.

Hopper, E. (2000). *Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods*. Ph.D. Thesis, University of Wales.

Hopper, E., & Turton, B. C. H. (2000). An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. *European Journal of Operational Research*, *128*, 34–57.

Hopper, E., & Turton, B. C. H. (2001). A review of the application of meta-heuristic algorithms to 2D strip packing problems. *Artificial Intelligence Review*, *16*, 257–300.

Iori, M., Martello, S., & Monaci, M. (2002). Metaheuristic algorithms for the strip packing problem. In P. Pardalos & V. Korotkich (Eds.), *Optimization and industry: new frontieres*. Norwell: Kluwer Academic.

Karabulut, K., & Inceoglu, M. M. (2004). A hybrid genetic algorithm for packing in 3D with deepest bottom left with fill method. In *Lecture notes in computer science* (Vol. 3261, pp. 441–450). Berlin: Springer.

Kenmochi, M., Imamichi, T., Nonobe, K., Yagiura, M., & Nagamochi, H. (2009). Exact algorithms for the 2-dimensional strip packing problem with and without rotations. *European Journal of Operational Research*, *198*, 73–83.

Kröger, B. (1993). *Parallele genetische Algorithmen zur Lösung eines zweidimensionalen Bin Packing Problems*. Ph.D. Thesis, Fachbereich Mathematik and Informatik, Universität Osnabrück.

Kröger, B. (1995). Guillotine bin packing: A genetic approach. *European Journal of Operational Research*, *84*, 645–661.

Lesh, N., Marks, J., McMahon, A., & Mitzenmacher, M. (2004). Exhaustive approaches to 2D rectangular perfect packings. *Information Processing Letters*, *90*, 7–14.

Lesh, N., Marks, J., McMahon, A., & Mitzenmacher, M. (2005). New heuristic and interactive approaches to 2D rectangular strip packing. *ACM Journal of Experimental Algorithmics*, *10*, 1–18.

Lodi, A., Martello, S., & Vigo, D. (1999). Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing*, *11*, 345–357.

Martello, S., & Vigo, D. (1998). Exact solution of the two-dimensional finite bin packing problem. *Management Science*, *44*, 388–399.

Martello, S., Monaci, M., & Vigo, D. (2003). An exact approach to the strip-packing problem. *INFORMS Journal on Computing*, *15*, 310–319.

Mumford-Valenzuela, C. L., Vick, J., & Wang, P. Y. (2004). Heuristics for large strip packing problems with guillotine patterns: an empirical study. In: *Metaheuristics: computer decision-making* (pp. 501–522). Norwell: Kluwer Academic.

Ortmann, F. G., Nthabiseng, N., & van Vuuren, J. H. (2010). New and improved level heuristics for the rectangular strip packing and variable-sized bin packing problems. *European Journal of Operational Research*, *203*, 306–315.

Pisinger, D. (2002). Heuristics for the container loading problem. *European Journal of Operational Research*, *141*, 382–392.

Riff, M. C., Bonnaire, X., & Neveu, B. (2009). A revision of recent approaches for two dimensional strip-packing problems. *Engineering Applications of Artificial Intelligence*, *22*, 823–827.

Schnecke, V. (1996). *Hybrid genetic algorithms for solving constrained packing and placement problems*. Ph.D. Thesis, Fachbereich Mathematik und Informatik, Universität Osnabrück.

Sixt, M. (1996). *Dreidimensionale Packprobleme. Lösungsverfahren basierend auf den Metaheuristiken Simulated Annealing und Tabu-Suche*. Ph.D. Thesis, Frankfurt am Main, Peter Lang, Europäischer Verlag der Wissenschaften.

Wäscher, G., Haussner, H., & Schumann, H. (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research*, *183*, 1109–1130.

Wei, L., Zhang, D., & Chen, Q. (2009). A least wasted first heuristic algorithm for the rectangular packing problem. *Computers & Operations Research*, *36*, 1608–1614.

Zhang, D., Liu, Y., Chen, S., & Xie, X. (2005). A meta-heuristic algorithm for the strip rectangular packing problem. In *Lecture notes in computer science, part III* (Vol. 3612, pp. 1235–1241). Berlin: Springer.

Zhang, D., Kang, Y., & Deng, A. (2006). A new heuristic recursive algorithm for the strip rectangular packing problem. *Computers & Operations Research*, *33*, 2209–2217.