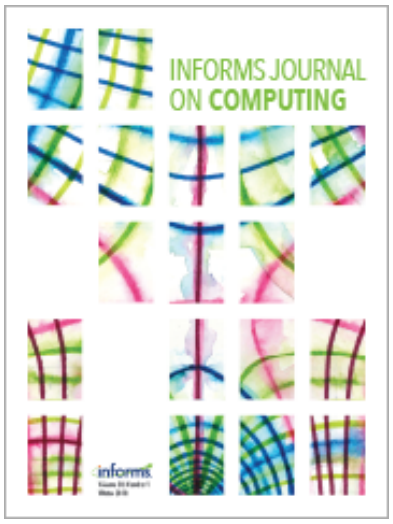


INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

A Tree Search Algorithm for Solving the Container Loading Problem

Tobias Fanslau, Andreas Bortfeldt,

To cite this article:
Tobias Fanslau, Andreas Bortfeldt, (2010) A Tree Search Algorithm for Solving the Container Loading Problem. INFORMS Journal on Computing 22(2):222-235. https://doi.org/10.1287/ijoc.1090.0338

INFORMS Journal on Computing
Vol. 22, No. 2, Spring 2010, pp. 222–235
ISSN 1091-9856 | EISSN 1526-5528 | 10 | 2202 | 0222

DOI 10.1287/ijoc.1090.0338

A Tree Search Algorithm for Solving the Container Loading Problem

Tobias Fanslau, Andreas Bortfeldt
Department of Information Systems, University of Hagen, 58084 Hagen, Germany
{tobias.fanslau@onlinehome.de, andreas.bortfeldt@fernuni-hagen.de}

This paper presents a tree search algorithm for the three-dimensional container loading problem (3D-CLP). The 3D-CLP is the problem of loading a subset of a given set of rectangular boxes into a rectangular container so that the packing volume is maximized. The method includes two variants: the full-support variant guarantees full support from below for all packed boxes, although this constraint is not taken into account by the nonsupport variant. The guillotine cut constraint is considered by both variants. The method is mainly based on two concepts. On the one hand, the block building approach is generalized. Not only are blocks of identical boxes in the same spatial orientation applied but also blocks of different boxes with small inner gaps. On the other hand, the tree search is carried out in a special fashion called a partition-controlled tree search. This makes the search both efficient and diverse, enabling a sufficient search width as well as a suitable degree of foresight. The approach achieves excellent results for the well-known 3D-CLP instances suggested by Bischoff and Ratcliff [Bischoff, E. E., M. S. W. Ratcliff. 1995. Issues in the development of approaches to container loading. *Omega* **23**(4) 377–390] with reasonable computing time.

Key words: container loading; 3D packing; heuristic; tree search
History: Accepted by David Woodruff, Area Editor for Heuristic Search and Learning; received December 2007; revised October 2008, March 2009; accepted March 2009. Published online in *Articles in Advance* July 29, 2009.

1. Introduction

Cutting and packing problems (C&P; Dyckhoff and Finke 1992) represent problems of the optimal use of resources. Whereas cutting problems are concerned with the best possible use of materials such as steel, glass, or wood, packing problems involve the best possible capacity utilization of packaging space. The effective utilization of material and transport capacities is of great economic importance in production and distribution processes. It also contributes to using natural resources economically, to limiting the growth in traffic, and, as a whole, to treating the environment with greater care. The development of even more effective cutting and packing methods remains an important task because, in view of the size of today's productions and distribution processes, even relatively minor growth in the utilization of material and space capacities can result in both considerable material savings and reductions in costs.

The subject of the present paper is the three-dimensional container loading problem (3D-CLP), which is formulated as follows: A large cuboid, the container, and a set of smaller cuboids, the boxes, are given. In general, the total volume of the boxes exceeds the container volume. A feasible arrangement of a subset of the given boxes is to be determined in such a way that the packed box volume is maximized, and where applicable, additional constraints are met. A box arrangement is deemed to be feasible if

—Each box lies completely in the container, i.e., does not exceed its boundary surfaces;

—No two boxes overlap; and

—Each box is parallel to the container's boundary surfaces.

A box type is defined in principle through the three side dimensions of a box. If there is only one box type in a box set, it is described as homogeneous. If there are only a few box types with a relatively large number of specimens per type, this is a weakly heterogeneous box set; on the other hand, if there are many box types with only a few examples of each type, the box set is strongly heterogeneous. This paper presents a tree search method for the 3D-CLP that is just as suitable for weakly heterogeneous box sets as for strongly heterogeneous. Both variants are explicitly differentiated in the new typology of the C&P problems (Wäscher et al. 2007). Using this typology, the method presented here can be applied to both the single large object placement problem (SLOPP, weakly heterogeneous) and to the single knapsack problem (SKP, strongly heterogeneous).

Three well-known constraints are considered:

(C1) *Guillotine cutting constraint*. All boxes that are stowed in a packing plan can be reproduced by a

series of guillotine cuts. The cutting area of a guillotine cut lies parallel to a boundary surface of the container, and the cut piece is always completely separated in two smaller parts.

(C2) *Support constraint*. The area of each box of a packing plan that is not placed on the floor of the container must be supported completely (i.e., 100%) by other boxes.

(C3) *Orientation constraint*. For certain box types, up to five of the maximal six possible orientations are prohibited.

Whereas the constraints (C2) and (C3) are of importance in practical packing scenarios (Bischoff and Ratcliff 1995), both the constraints (C1) and (C3) are relevant in 3D cutting: often, automated cutting machines can only perform guillotine cuts, whereas an orientation constraint occurs frequently if the items to be cut are decorated or corrugated. In this paper a tree search method with two variants is introduced: whereas the full-support variant observes the support constraint (C2), this is not the case with the nonsupport variant; constraints (C1) and (C3) are fulfilled by both variants. A comparison of the results of both variants reveals the considerable influence of the required full box support on the full use of space as the primary optimization objective. The terminology of packing will be used throughout, although the proposed method can also be applied in a cutting context.

The rest of the paper is divided as follows. Section 2 provides an overview of the literature, and §3 presents the tree search method. Section 4 is dedicated to computational experiments, and §5 summarizes the paper.

2. Literature Overview

The 3D-CLP is NP-hard in a strict sense and is also regarded in practice as extremely difficult (Pisinger 2002). Up to now, only a few exact methods were suggested. Fekete and Schepers (1997) develop a general framework for the exact solution of multidimensional packing problems. Martello et al. (2000) present an exact branch-and-bound (B&B) method for the 3D-CLP. This is part of a method for the three-dimensional bin packing problem with which instances with up to 90 boxes could be solved.

Heuristic methods without an optimality guarantee for the 3D-CLP can be divided into three groups with regard to the method type:

(1) *Conventional heuristics*. These include construction methods (e.g., greedy algorithms) and improvement methods (e.g., local search algorithms). Conventional heuristics for solving the 3D-CLP were suggested, e.g., by Bischoff et al. (1995), Bischoff and Ratcliff (1995), and Lim et al. (2003).

(2) *Metaheuristics*. In the last 10 years, metaheuristic search strategies constituted the preferred method types for the 3D-CLP. Genetic algorithms (GAs) were suggested among others by Hemminki (1994), Gehring and Bortfeldt (1997, 2002), and Bortfeldt and Gehring (2001). Tabu search algorithms (TSs) came from Sixt (1996) and Bortfeldt et al. (2003). Simulated annealing methods (SAs) were developed by Sixt (1996) and by Mack et al. (2004). A method of local search based on the Nelder and Mead approach dates from Bischoff (2004). Moura and Oliveira (2005) as well as Parreño et al. (2007) introduce a greedy randomized adaptive search procedure (GRASP).

(3) *Tree search methods*. Incomplete tree search or graph search methods (in brief TRS) were applied successfully to the 3D-CLP. Mention should be made of the method of the And/Or graph search by Morabito and Arenales (1994), the tree search methods from Eley (2002) and Hifi (2002), and the B&B method from Terno et al. (2000) (the integrated CLP method is meant) and Pisinger (2002).

The existing 3D-CLP methods are based on different heuristic packing approaches that determine the structure of generated packing plans (cf. Pisinger 2002):

(1) *Wall building approach*. The container is filled by vertical cuboid layers ("walls") that mostly follow the longest side of the container. The approach is realized, among others, in Bortfeldt and Gehring (2001) and in Pisinger (2002).

(2) *Stack building approach*. The boxes are packed in a suitable manner in stacks, which are arranged on the floor of the container in a way that saves the most space. Characteristic of this approach is that the stacks do not themselves form walls as defined before. Advocates of this approach are, among others, the heuristic from Bischoff and Ratcliff (1995) and the GA from Gehring and Bortfeldt (1997).

(3) *Horizontal layer building approach*. The container is filled from bottom to top through horizontal layers that are each intended to cover the largest possible part of the (flat) load surface underneath. This approach is realized in the methods from Bischoff et al. (1995) and Terno et al. (2000).

(4) *Block building approach*. The container is filled with cuboid blocks that mostly contain only boxes of a single type with the same spatial orientation. The approach is related to approach (3), but the main motive here is to fill the largest-possible subspaces of the container without internal losses; compare as well the characterization of this approach in Eley (2002). Representatives of the approach are the TS method from Bortfeldt et al. (2003), the tree search method from Eley (2002), and the SA/TS hybrid, respectively, from Mack et al. (2004).

(5) *Guillotine cutting approach*. This approach is based on a slicing tree representation of a packing

plan. Each slicing tree corresponds to a successive segmentation of the container into smaller pieces by means of guillotine cuts, whereby the leaves correspond to the boxes to be packed. Morabito and Arenales's graph search method (1994) is based on this approach.

The great majority of the methods mentioned observe the support constraint (C2) and the orientation constraint (C3) as well. A considerable part of the listed methods also include further constraints from the packing context in the problem, e.g., a weight constraint for the freight; compare among others Terno et al. (2000), Bortfeldt and Gehring (2001), Eley (2002), and Mack et al. (2004). The heuristic developed by Bischoff et al. (1995) for loading pallets and the GRASP method from Moura and Oliveira (2005) prove to be particularly sensitive methods for the stability of the load. Bischoff (2004) considers a complicated overstacking constraint, which limits the pressure applied to the surface covering of boxes.

Up to now only a few contributions refer explicitly to the cutting application context of the 3D-CLP (Hifi 2002). Other contributions present methods that also comply with the guillotine cut constraint (C1); compare among others Morabito and Arenales (1994), Bortfeldt and Gehring (2001), and Pisinger (2002). Mack et al. (2004) examine, to an extent, the influence of the support constraint (C2) on the achievable volume utilization.

3. The Tree Search Algorithm

The proposed tree search method is referred to as CLTRS (container loading by tree search). It is based in the main on two concepts:

—The block building approach is generalized. Along with the usual compact blocks from identical boxes with the same spatial orientation, blocks with small inner gaps are now also used to construct packing plans.

—A special form of tree search tries to keep the search effort low and, at the same time, ensures a suitable balance between search diversity and foresight during the search.

The following presentation is focused on the core concepts, whereas a more detailed description of the method can be found in Fanslau and Bortfeldt (2008). Differences between the nonsupport and the full-support variants of CLTRS will be discussed at appropriate points.

3.1. Basic Concepts and Overall Algorithm

First some basic concepts are introduced and terminological agreements are made.

Let the container be embedded in the first octant of a 3D coordinates system (3D-CS). This is shown in Figure 1, which also illustrates terms such as "left,"

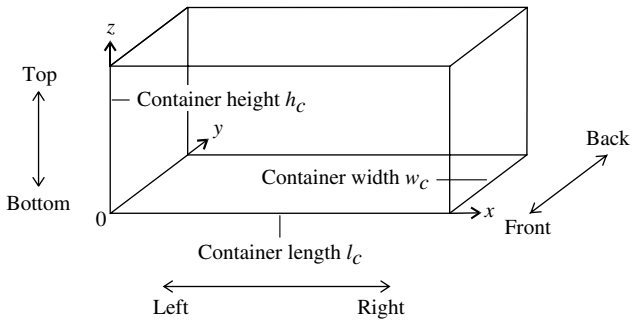


Figure 1 Container in 3D Coordinates System

"in front of," etc. A cuboid (box, empty space in the container, envelope cuboid of a box arrangement) is referred to as oriented if it may no longer be turned with regard to the 3D-CS (but may possibly still be displaced). Let it be agreed that an oriented cuboid always lies "somewhere" in the first octant of the 3D-CS and parallel to its axes. The reference corner of an oriented cuboid is then understood as being the corner nearest to the origin of the 3D-CS. The dimensions of an oriented cuboid in the coordinate directions are designated with *mx*, *my*, and *mz*, respectively.

A residual space is an oriented cuboid space in the container given by its three side dimensions and its reference corner. During the search, residual spaces are constructed continuously and later filled by boxes where possible. Residual spaces that are already constructed but have not yet been processed are managed in a residual space stack.

A block is an arrangement made up of one or more oriented boxes. During the search, a block is represented above all through the dimensions of the oriented envelope cuboid of the arrangement and by the packed box set. However, the block data also clearly stipulate the spatial orientation and (relative) position of all boxes within the block.

The container is filled through the successive placing of blocks. To place a block means to arrange it in the reference corner of a residual space. Figure 2 shows the placing of a block, represented by its envelope cuboid, in a residual space.

The search state summarizes the situation after some blocks have been placed. It contains the following components: a (generally incomplete) packing plan including some block placements, the set of free (still not packed) boxes, and the stack of residual spaces.

The overall algorithm of the method is shown in Figure 3. The total search is divided into two stages. For each stage, a specific block list that contains all blocks that can be placed during the stage is generated first. In stage 1, only blocks of identical boxes in the same spatial orientation that together form a

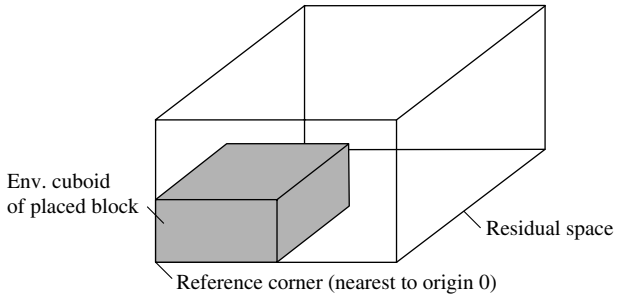


Figure 2 A Residual Space with a Placed Block

complete cuboid are provided. In stage 2, blocks are generated whose box arrangements usually do not completely fill up their envelope cuboid. Although the block list for stage 1 is tailored to weakly heterogeneous box sets, the block list in stage 2 proves to be particularly suitable for strongly heterogeneous box sets. However, in many cases the best utilization for weakly heterogeneous instances as well is achieved with the block list of the second stage. For this reason, a two-phase search approach is selected to diversify the search.

Within each stage, one complete packing plan after the other is generated until the given time limit specific for each stage is exceeded. Finally, the best packing plan generated over both stages is output.

The generation of a complete packing plan in a stage is referred to as iteration. At the start of an iteration, the search state is initialized. The residual space stack then contains the container as the sole residual space, the current solution is empty, and all boxes are still free. Following this, the uppermost residual space of the stack is processed until the stack is empty and the next solution is thus complete. To fill a residual space, a suitable block is selected from the current block list where possible. Afterwards, the search state is updated. (If no block fits into the residual space, it is only removed from the stack.)

```
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,
      free 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) exceeded)
endfor
output best packing plan over both stages
```

Figure 3 Overall Algorithm of Method CLTRS

To find the best block for the current residual space, a tree search is carried out with an effort defined through the *search_effort* internal parameter. This is doubled from solution to solution, and thus better and better packing plans can be calculated. To a certain extent, CLTRS behaves like a construction heuristic in each iteration: if a block has been specified, this decision is not revised when further blocks are determined.

3.2. The Generalized Block Building Approach

Up to now, good results have only been achieved by means of the block building approach for weakly heterogeneous instances. To also make the block-building approach fruitful for the strongly heterogeneous case, it has to be generalized. Although blocks are to fill relatively large volumes without loss, or practically without loss, they may now also consist of boxes of several types. The generalization is brought about through the introduction of new block types, which will be explained in detail below.

3.2.1. Simple Blocks. Blocks in the traditional sense are referred to as simple blocks. A simple block is a cuboid arrangement from $nx \times ny \times nz$ boxes of the same type in the same spatial orientation, where nx boxes lie in the x direction, ny boxes in the y direction, and nz boxes in the z direction.

3.2.2. General Blocks for the Nonsupport Variant. A general block for the nonsupport variant is also referred to as a general nonsupport block (GNS block). It contains an arrangement of one or more boxes and the appropriate oriented envelope cuboid C_{env}. The envelope cuboid is touched on each of its inner sides by the box arrangement as usually but it is generally not filled completely by boxes.

General nonsupport blocks result if single boxes or smaller GNS blocks are arranged next to each other as shown in Figure 4. They can be defined recursively as follows:

(i) A single box of any type in any feasible spatial orientation is a GNS block.

(ii) Let bl_1 and bl_2 be GNS blocks. The envelope cuboids $C_{\text{env}}(bl_1)$ and $C_{\text{env}}(bl_2)$ of both GNS blocks with the enclosed boxes are arranged contiguous in x direction or contiguous in y direction or contiguous in z direction (see Figure 4). Each of the three resulting combined box arrangements bl_c forms a GNS block with the appropriate envelope cuboid $C_{\text{env}}(bl_c)$ if the following conditions are fulfilled:

(ii.1) *Box availability*. The box set of arrangement bl_c is a subset of the given stock of boxes.

(ii.2) *Space availability*. The envelope cuboid $C_{\text{env}}(bl_c)$ can be arranged completely in the given container.

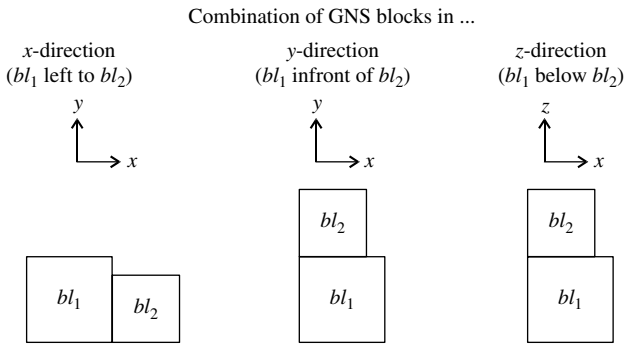


Figure 4 Combining GNS Blocks in *x*-, *y*-, and *z*-Direction

(ii.3) *Contact surfaces comparison*. If the GNS blocks bl_1 and bl_2 are combined in x-direction, bl_1 may only lie to the left of bl_2 if the following applies for the contact surfaces:

$$my(C_{\text{env}}(bl_1)) \times mz(C_{\text{env}}(bl_1)) \geq my(C_{\text{env}}(bl_2)) \times mz(C_{\text{env}}(bl_2)).$$

Similar conditions hold if bl_1 and bl_2 are combined in y-direction or in z-direction.

(ii.4) *Space utilization*. The percent space utilization of the envelope cuboid $C_{\text{env}}(bl_c)$, given by the quotient $fr :=$ (volume of all boxes of bl_c)/(volume of $C_{\text{env}}(bl_c)$)$\times$100, does not fall below the given minimal space utilization *min_fr* (in percent); i.e., $fr \geq min\_fr$.

(iii) Only a box arrangement, which is identifiable in accordance with (i) and (ii) as a GNS block, represents a GNS block. □.

The definition of a GNS block may be commented on as follows: the infringement of the conditions (ii.1) and (ii.2) would lead to blocks that are of no use for the search. The contact surfaces condition (ii.3) stipulates which of the two original blocks lies in the reference corner of a combined block bl_c and helps to avoid redundant GNS blocks. Condition (ii.4) is decisive: if the parameter *min_fr* selected for block generation is large enough (for example, *min_fr* = 98%), only blocks that are practically without loss are generated, whereas at the same time, the combination of boxes of different types in GNS blocks is not ruled out. The definition therefore conforms to the above-mentioned requirements for a generalization of the original block concept. In addition, it is obvious that every simple block is a GNS block.

3.2.3. General Blocks for the Full-Support Variant. Because GNS blocks generally do not satisfy the support constraint (C2), another block type is necessary for the full-support variant. A general block for the full-support variant is also referred to as a general full-support block (GFS block). Together

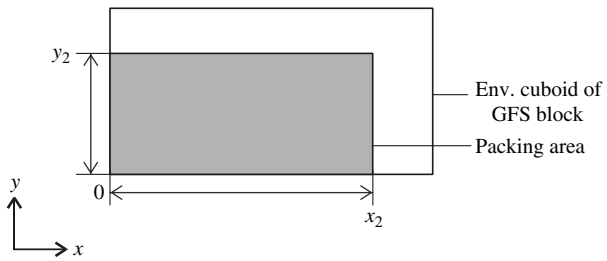


Figure 5 GFS Block with Packing Area

with a box arrangement and the appropriate envelope cuboid C_{env}, a GFS block contains a packing area (see Figure 5). This one is part of the top area of the envelope cuboid with the following characteristic: if a box lies completely on the packing area, its area is supported completely by other boxes. It is required that the packing area is always rectangular and its front left-hand corner (i.e., the corner nearest to the origin of the 3D-CS) is incidental to the front left-hand corner of the top area of the envelope cuboid. Consequently, the packing area of a GFS block is clearly determined by the coordinates x_2, y_2 of its rear right-hand corner relative to the reference corner of the envelope cuboid.

General full-support blocks are defined (and constructed) analogously to general nonsupport blocks. Although parts (i) and (iii) of the above definition can be adopted, part (ii) is to be extended by two conditions. The height condition requires that in a combination of two GFS blocks bl_1 and bl_2 in a horizontal direction (x or y), the heights of the envelope cuboids of bl_1 and bl_2 coincide. The packing area condition guarantees that the packing area of a new, combined GFS block is also rectangular and starts in the front left-hand corner of the envelope cuboid's top area (see Fanslau and Bortfeldt 2008 for further details).

All in all, GFS blocks stand for a generalization of the original block concept that takes the support constraint (C2) into account. Note that each simple block is also a GFS block whose packing area coincides with the top area of the block.

3.2.4. Generating Blocks. Simple blocks are generated at the start of the first stage of the search and general blocks are made available in the block list at the start of the second stage. The nonsupport and the full-support variant differ with regard to general blocks (GNS blocks versus GFS blocks).

The block generation may be characterized as follows. Starting from individual boxes, the blocks for the different method variants and stages of the search are generated systematically on the basis of the recursive definitions shown above. If the envelope cuboid and the packed box sets of two blocks coincide, only one of these blocks is included in the block list to reduce the redundancy of the search. The number of

places in the block list is restricted by the parameter *max_bl*. Hence, the block generation is interrupted once *max_bl* blocks have been included in the block list for a search stage. The parameter values *max_bl* and the minimal block utilization *min_fr* are determined experimentally. The block list is sorted in descending order according to the block volume.

3.3. Determining the Best Block for a Residual Space

Let a search state s be given with an incomplete solution including i ($i \geq 0$) placed blocks. To determine a block for the uppermost residual space *rstop* of the residual space stack, starting from s—i.e., retaining the i blocks already placed—different complete solutions are generated experimentally. Obviously, the $(i+1)$th block of all temporarily generated solutions fills the residual space *rstop*. Finally, the $(i+1)$th block of the complete temporary solution with maximal packed volume is returned as the best block for *rstop*. A multiple-tree search is carried out to generate complete solutions starting each time from search state s. This tree search is described in a bottom-up fashion in the sequel.

3.3.1. Basic Search Phases. A basic search phase transforms a given search state s' with j ($j \geq 0$) placed blocks into a search state s'' with at most $j+d$ placed blocks where d ($d \geq 0$) is a given depth parameter. A tree of states with root s' and depth $d+1$ is constructed. The (maximal) number of successors per state *ns* is calculated in accordance with

$$ns = \max\{min\_ns, \max\{q \mid q \geq 0 \text{ and } q^d \leq search\_effort\}\}. \quad (1)$$

For example, for $min\_ns = 2$, $search\_effort = 4$, and $d = 2$, one has $ns = 2$. The parameter *min_ns* indicates the minimal number of successors per search state throughout the search.

The *ns* successors of a state s_c are generated as follows. The uppermost residual space in the stack of s_c is filled alternately with one of the *ns* largest appropriate blocks. Afterwards, state s_c is updated each time to get the belonging successor state. A block is appropriate if and only if it fits into the residual space and its boxes are still free. If an uppermost residual space cannot be filled by any block of the current block list, then the next residual space is filled, etc.

The states of depth $d+1$ generally contain still incomplete solutions. Therefore, a state s_c of depth $d+1$ is further expanded (where possible) by means of the so-called greedy completion: starting from s_c, the uppermost residual space is repeatedly filled with an appropriate block with maximal volume ($ns = 1$) until a state with a complete solution is achieved.

Finally, the best generated complete solution with $j+d+x$ placed blocks and with maximal packed volume determines the search state s'' with $j+d$ blocks that is returned. In other words, s'' is the state with $j+d$ placed blocks from which the best complete solution (generated in this phase) results by greedy completion. Moreover, the best complete solution of the whole search is updated by the best complete phase solution where necessary.

A basic search phase can be written as an operator (unique map) $E(d)$, and transforming a state s' into a state s'' then reads $s'' = E(d)\ (s')$.

3.3.2. Chaining of Basic Search Phases and Partition Searches. Two basic search phases, $E(d_1)$ and $E(d_2)$, can be combined by chaining: a given search state s_0 with j placed blocks is first transformed by means of $E(d_1)$, resulting in a search state s_1 with $j+d_1$ placed blocks. Afterwards, the output state s_1 of $E(d_1)$ is taken as input state for the second search phase $E(d_2)$, and this results in a search state s_2 with $j+d_1+d_2$ placed blocks. The combined operator is written as $E(d_2)\ \mathrm{o}\ E(d_1)$, and for a given initial state s_0 the whole transformation reads $s_2 = E(d_2)\ \mathrm{o}\ E(d_1)\ (s_0)$. Obviously, more than two basic search phases can be combined in a similar way, and the meaning of the combined operator is clear from its phases and the selected order of these phases.

An ordered partition π of a whole number n ($n \geq 1$) is given by a $(l+1)$-tuple of integers:

$$\pi = (l, d_1, \ldots, d_l), \quad \text{with } l \geq 1,\ d_j \geq 0,\ j = 1, \ldots, l, \text{ and } \sum_{j=1}^{l} d_j = n. \quad (2)$$

For a given integer n, each ordered partition defines a partition search $E(\pi)$ according to

$$E(\pi) = E'(d_l)\,\mathrm{o}\,E(d_{l-1})\,\mathrm{o}\cdots\mathrm{o}\,E(d_2)\,\mathrm{o}\,E(d_1). \quad (3)$$

Following definition (3), a partition search corresponds to a combined operator as introduced above, and it is composed of some basic search phases where the phases and their order are given by an ordered partition of an integer n.

Note that the meaning of the last-phase operator $E'(d_l)$ is slightly modified (marked by a stroke): whereas the "normal" phase operator $E(d_l)$ provides a state s'' with an incomplete solution, the modified operator $E'(d_l)$ provides a state that results by greedy completion from s'' (see §3.3.1). Therefore, a partition search always returns a state with a complete solution.

The integer n is called total depth *td* in what follows. Obviously, the total depth *td* indicates the total number of blocks that are added to a given initial state by a partition search (if the final greedy completion is neglected).

3.3.3. Partition-Controlled Tree Search. Now, the calculation of the best block for the uppermost residual space of a given search state s with i placed blocks can be formulated as follows:

(i) Define total depth td according to

$$td = \max\{1, \max\{d \mid d \geq 0 \text{ and } min\_ns^d \leq search\_effort\}\}. \quad (4)$$

(For example, for $min\_ns = 2$ and $search\_effort = 4$, the total depth is $td = 2$.)

(ii) For each ordered partition π of td, apply the corresponding partition search $E(\pi)$ to state s and generate state $E(\pi)$ (s) containing a complete solution.

(iii) Return the $(i+1)$th block of the best complete solution with maximal packed volume.

Clearly, the parameter *search_effort* rules the extent of the search because the total depth td, the number of partitions, and the number of successors per state ns depend on it. In both search stages, the parameter *search_effort* is doubled several times (see Figure 3), and by this, more and more complete solutions are generated experimentally to find the (final) block for a residual space; i.e., better and better solutions can be achieved.

The presented schema of a multiple-tree search together with the mechanism of an iterative enlargement of the search effort spent is called partition-controlled tree search (PCTRS).

3.3.4. Illustrative Example. PCTRS is illustrated by the following example. Let $min\_ns = 2$, $search\_effort = 8$ (fourth iteration of a search stage); hence, total depth $td = 3$. Consequently, separate tree searches must be carried out for the following partitions: (1) $\pi_1 = (1, 3)$, (2) $\pi_2 = (2, 2, 1)$, (3) $\pi_3 = (2, 1, 2)$, and (4) $\pi_4 = (3, 1, 1, 1)$. Figure 6 illustrates the four partition searches.

Resulting states of basic search phases (above denoted by s'') are written as $sbest_j$, whereas corresponding states with complete solutions that result from greedy completion are denoted by $sbest_{jc}$ ($j = 1, 2, 3$). As defined above, each partition search returns the state $sbest_{lc}$ with a complete solution (l denotes the length of a partition as before).

3.3.5. Characteristics of Partition-Controlled Tree Search. Some essential characteristics of PCTRS are subsequently highlighted.

Only two parameters, namely, the minimal number of successors per state *min_ns* and a time limit, are needed to control the tree search. Note that all other parameters (*search_effort*, total depth td, etc.) are no input values.

PCTRS can be viewed as a relatively economical kind of tree search, as the previously shown example reveals. In the example under consideration, complete solutions are to be generated through a tree search with a total depth $td = 3$. If a tree search with a constant number of successors $ns = 8$ is carried out instead, then $8^3 = 512$ complete solutions are to be generated. It is assumed that the search with eight successors per state is performed over three depth levels only, and afterwards, greedy completion is applied to each incomplete solution. In contrast, only $8 + 12 + 12 + 24 = 56$ complete solutions are generated with PCTRS; this is only 11% of the solutions generated with constant $ns = 8$ (see Figure 6). The value $ns = 8$ was taken for the comparison because it is the highest number of successors used in the PCTRS example, and this value was used in 6 (i.e., 40%) of all 15 state expansions in the four partition searches.

The number of successors ns of a basic search phase determines the (local) search width or diversity. The search depth d_j stipulates at the same time the number of consecutive search levels that are considered in the decision made by the search phase and in this way determines its degree of foresight. In the above example, the only search phase of partition π_1 has a greater degree of foresight with search depth $d_1 = 3$ than the three phases of π_4 each with the search depth 1. Both quantities, the number of successors per state and the number of simultaneous considered depth levels, are key factors of a successful search. Roughly speaking, the greater the number of successors and the depth of a basic search phase, the better the solution quality to be expected. High values for both factors will result in a search effort that is no more reasonable. Therefore, the depth d and the number of successors ns of basic search phases are defined in such a way that a higher value for one factor is accompanied by a lower value for the other factor (for a constant value of *search_effort*; see Equation (1)).

With the PCTRS a separate search is carried out for each partition of a given total depth td. Consequently, many different combinations of search width and degree of foresight are examined within individual basic search phases. Moreover, basic search phases are combined in different ways; e.g., in the partitions π_2 and π_3 the same phases are combined in different order. All in all, it seems that PCTRS is able to handle the success factors of search width and degree of foresight in a flexible and balanced way.

3.4. Management of Residual Spaces

The management of residual spaces forms another building block of method CLTRS. Here it is only sketched, whereas a full description of the space management can be found in Fanslau and Bortfeldt (2008). If a block bl was placed in a residual space rs, this old space is removed from the residual space stack. The remaining free space in rs is cut without any waste into three new residual spaces that are then inserted in the stack.

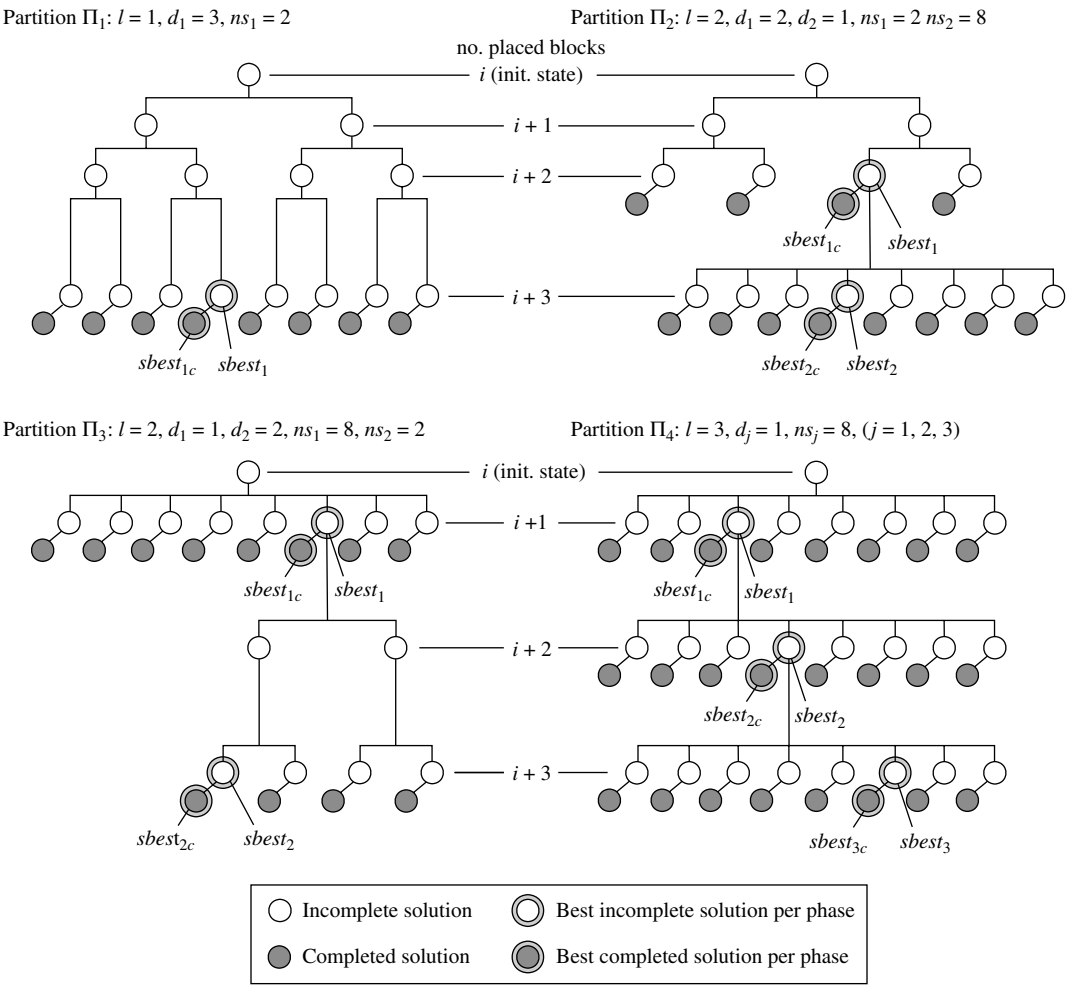


Figure 6 Partition Search with Four Different Partitions

Figure 7 shows residual space *rs* and block *bl*, each with their dimensions and the differences $rmx = rsx - blx$, $rmy = rly - bly$, and $rmz = rsz - blz$ called reduced measures.

For each waste-free cutting variant, one new residual space (*nrs_x*) has the *x*-dimension *rmx*, a further new space (*nrs_y*) has the *y*-dimension *rmy*, whereas the third new space (*nrs_z*) has the *z*-dimension *rmz*. Moreover, exactly one of the new residual spaces coincides with the residual space *rs* in two dimensions and is designated as maximal. Exactly one of the new spaces coincides with the block *bl* in two dimensions and is designated as minimal, whereas the third space is designated as medium. The three residual spaces are always generated by three guillotine cuts, the first of which subdivides the old residual space *rs*.

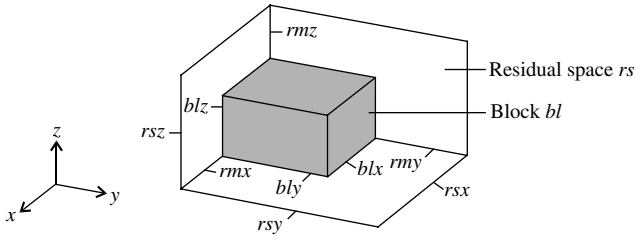
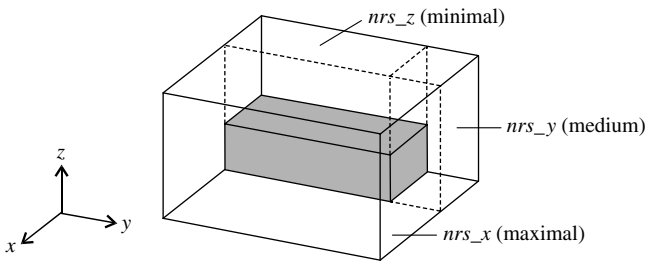


Figure 7 Residual Space *rs*, Block *bl*, and Reduced Measures

The reduced measures are the critical dimensions of the new residual spaces; i.e., they can above all have the effect that new residual spaces can no longer be filled. If new residual spaces would pair a small reduced measure with other, relatively large, residual space measures, large volume losses would be risked. Therefore, new residual spaces are cut in such a way that larger reduced measures are combined with other larger, residual space measures.

For example, assuming that $rmx \geq rmy \geq rmz$ and that all reduced measures can be filled by at least one box, then the three new spaces are cut as shown in Figure 8.

Note that the maximal reduced measure *rmx* belongs to the maximal new space (in the above-defined sense), whereas the minimal reduced measure *rmz* belongs to the minimal new space.



	Dimensions		
Space	*mx*	*my*	*mz*
nrs_x	*rmx*	*rsy*	*rsz*
nrs_y	*blx*	*rmy*	*rsz*
nrs_z	*blx*	*bly*	*rmz*

Figure 8 Residual Spaces of a Cutting Variant

Inserting new residual spaces into the residual space stack takes into account that sometimes a transfer of unused space between different residual spaces is desirable. Assuming that all three new spaces are large enough to accommodate further boxes, they are inserted in the following order: minimal space, maximal space, medium space (in the sense defined above). In this order it can be guaranteed that packing space can be transferred later to the minimal new residual space. Let, for example, residual space *rs* be cut as shown in Figure 8. If the maximal residual space *nrs_x* now proves to be no more fillable when it is processed, the minimal residual space *nrs_z* can be enlarged by setting its *x*-dimension to *rsx*. Thus, it may still be possible to use the cuboid with the dimensions $mx = rmx$, $my = bly$, and $mz = rmz$ that is transferred from *drs_x* to *drs_z*.

Above, the space management was sketched for the nonsupport variant. The management is adapted to the full-support variant by some simple modifications. For example, only cutting variants that avoid overhanging boxes are admissible.

3.5. Handling the Selected Constraints

Finally, handling the selected constraints (C1) to (C3) is dealt with briefly.

The guillotine cut constraint (C1) fulfills itself in both the nonsupport variant and in the full-support variant of the method. The boxes of any block can obviously be reproduced by guillotine cuts. Each placed block lies fully in a residual space and finally, starting from the container, additional residual spaces are created only through guillotine cuts.

The support constraint (C2) is fulfilled in the full-support variant. On the one hand, with this variant only blocks are formed in which all boxes that do not lie on the floor of the block are supported 100% from below; on the other hand, residual spaces above blocks are constructed in such a way that no overhanging boxes can occur.

In addition, and in order to study the impact of a specific required support of boxes on the volume utilization, the parameter *min_support* is introduced. If a box is not placed on the container floor, the percentage *min_support* of its base area is to be supported by other boxes. Of course, the variants nonsupport and full-support correspond to $min\_support = 0$ and $min\_support = 100$, respectively. In all other cases ($0 < min\_support < 100$), the management of residual spaces is adopted from the full-support variant. The same holds in principle for the generation of blocks. However, if general blocks are in demand and two partial blocks are stacked one upon the other, then the upper partial block may overhang in one horizontal direction. Nevertheless, each box of a block has to be supported from below in accordance with the given *min_support* value. In this way, the full-support condition is softened and more general blocks can be produced.

The orientation constraint (C3) is complied with in that unfeasible box orientations are avoided per box type and block.

4. Computational Experiments and Results

The CLTRS method was implemented in C, and multiple computational experiments were carried out.

The 1,600 3D-CLP instances from Bischoff and Ratcliff (1995) and from Davies and Bischoff (1998) were used for the test; below they are referred to as Bischoff-Ratcliff instances or BR instances, respectively. The 1,600 BR instances are subdivided into 16 test cases with 100 instances each that are numbered BR0 to BR15. The 100 instances of each test case coincide in the number of box types. The box sets of the different test cases vary from homogeneous through weakly heterogeneous to strongly heterogeneous; details can be found in Table 1. Each instance includes the orientation constraint (C3), which prohibits the use of one or two larger side dimensions as the height dimension. With the test of both CLTRS variants—nonsupport and full-support—the guillotine cut constraint (C1) is satisfied. Only with the test of the full-support variant is it presupposed that the support constraint (C2) has to be complied with in addition.

Two different PCs were used for the calculations: the first one with an AMD Athlon processor (64 FX60, 2.6 GHz) with 2,048 MB RAM, and the second one

Table 1 Results of CLTRS and Other Methods for BR Instances (100% Support *Not* Required)

Test case	No. of box types	Avg. no. of boxes per type	Lim et al. (2003)	Bortfeldt et al. (2003)	Mack et al. (2004)	Parreño et al. (2007)	CLTRS nonsupport (set A)	CLTRS nonsupport (set B)
BR0	1	206	—	—	—	—	**89.95**	**89.93**
BR1	3	50	88.70	93.52	94.41	93.85	**95.05**	**94.65**
BR2	5	27	88.17	93.77	93.82	94.22	**95.43**	**94.80**
BR3	8	17	87.52	93.58	94.02	94.25	**95.47**	**94.77**
BR4	10	13	87.58	93.05	93.68	94.09	**95.18**	**94.49**
BR5	12	11	87.30	92.34	93.18	93.87	**95.00**	**94.09**
BR6	15	9	86.86	91.72	92.64	93.52	**94.79**	**93.93**
BR7	20	7	87.15	90.55	91.68	92.94	**94.24**	**93.34**
BR8	30	4	—	—	—	91.02	**93.70**	**92.68**
BR9	40	3	—	—	—	90.46	**93.44**	**92.49**
BR10	50	3	—	—	—	89.87	**93.09**	**92.04**
BR11	60	2	—	—	—	89.36	**92.81**	**91.75**
BR12	70	2	—	—	—	89.03	**92.73**	**91.42**
BR13	80	2	—	—	—	88.56	**92.46**	**91.08**
BR14	90	1	—	—	—	88.46	**92.40**	**90.80**
BR15	100	1	—	—	—	88.36	**92.40**	**90.67**
Mean BR1–BR7			87.6	92.7	93.2	93.8	**95.0**	**94.3**
Mean BR1–BR15			—	—	—	91.5	**93.9**	**92.9**
Mean time BR1–BR7			—	121	222	302	319	51
Mean time BR1–BR15			—	—	—	238	320	53
CPU frequency (MHz)			600	$4 \times 2{,}000$	$4 \times 2{,}000$	1,500	2,600	800

Note. Computing times are in seconds, best results appear in bold, and there is no comparison between CLTRS runs.

with an Intel Pentium III processor (800 MHz) with 256 MB RAM. The weaker PC serves for comparisons with older (earlier published) methods, whereas the stronger PC yields better solutions for given time limits. There are two parameter sets. In set A, the parameters were set as follows: maximal number of blocks $max\_bl = 10{,}000$, minimal space utilization for blocks $min\_fr = 98\%$, minimal search width $min\_ns = 3$, and time limits for stages 1 and 2: 60 and 180 seconds, respectively. In parameter set B, only the time limits were changed to 10 seconds (stage 1) and 30 seconds (stage 2). Moreover, the parameter sensitivity was examined, as reported later.

In the sequel, CLTRS is compared to other methods (§4.1), the core concepts of CLTRS are evaluated (§4.2), and other issues such as loading stability and parameter sensitivity are dealt with (§4.3).

4.1. Comparison to Other Methods

Because there is empirical evidence for a strong impact of support constraint (C2) on the volume utilization (as discussed at the end of this section), the comparison of CLTRS to other methods from the literature is organized in two parts. In the first part, the nonsupport CLTRS variant is compared only to methods that also do *not* guarantee full support of packed boxes (see Table 1), whereas in the second part the full-support CLTRS variant is compared only with methods that satisfy constraint (C2) as well (see Table 2).

Both tables are organized similarly. For each method and each BR test case (of 100 instances), the mean volume utilization as a percentage of the container volume is indicated if available. The volume utilizations and the computing times (in seconds) are averaged over the 7 test cases BR1 to BR7 and (if possible) over the 15 test cases BR1 to BR15 as well. Note that up to now there have been only a few results for test case BR0. For each method, the cycle frequency of PC(s) used for calculations is added (if known); for the types of the compared methods, see §2. In both Tables 1 and 2 there are two columns for the appropriate CLTRS variant. In the first column, results are presented for the 2.6 GHz PC and parameter set A. In the second column, results are indicated for the 800 MHz PC and parameter set B.

If full support of boxes is *not* required, then method CLTRS achieves the best mean volume utilizations for all 15 test cases BR1 to BR15 (a comparison for BR0 is not possible). Obviously, the second-best results were generated by the GRASP method from Parreño et al. (2007), and the comparison is focused on this algorithm. Only the best results of the GRASP method are quoted here, and the mean computing time over the 15 test cases BR1 to BR15 is adjusted accordingly (cf. Parreño et al. 2007, Tables 5 and 6). If the CLTRS calculations with the 2.6 GHz PC and parameter set A are viewed, the mean improvement compared to the GRASP algorithm amounts to 2.4% points. Whereas the improvement is only 1.2% points averaged over the test cases BR1 to BR7, there is a mean improvement of 3.5% points for the test cases BR8 to BR15. Hence, the dominance is much stronger for strongly heterogeneous box sets.

Table 2 Results of CLTRS and Other Methods for BR Instances (100% Support Required)

Test case	Terno et al. (2000)	Bortfeldt and Gehring (2001)	Gehring and Bortfeldt (2002)	Eley (2002)	Bischoff (2004)	Moura and Oliveira (2005)	CLTRS full-support (set A)	CLTRS full-support (set B)
BR0	—	—	—	—	—	—	**89.83**	**89.82**
BR1	89.9	87.81	88.10	88.00	89.39	89.07	**94.51**	**94.26**
BR2	89.6	89.40	89.56	88.50	90.26	90.43	**94.73**	**94.36**
BR3	89.2	90.48	90.77	89.50	91.08	90.86	**94.74**	**94.30**
BR4	88.9	90.63	91.03	89.30	90.90	90.42	**94.41**	**93.94**
BR5	88.3	90.73	91.23	89.00	91.05	89.57	**94.13**	**93.50**
BR6	87.4	90.72	91.28	89.20	90.70	89.71	**93.85**	**93.23**
BR7	86.3	90.65	91.04	88.00	90.44	88.05	**93.20**	**92.49**
BR8	—	89.73	90.26	—	—	86.13	**92.26**	**91.54**
BR9	—	89.06	89.50	—	—	85.08	**91.48**	**90.69**
BR10	—	88.40	88.73	—	—	84.21	**90.86**	**89.98**
BR11	—	87.53	87.87	—	—	83.98	**90.11**	**89.24**
BR12	—	86.94	87.18	—	—	83.64	**89.51**	**88.74**
BR13	—	86.25	86.70	—	—	83.54	**88.98**	**88.01**
BR14	—	85.55	85.81	—	—	83.25	**88.26**	**87.27**
BR15	—	85.23	85.48	—	—	83.21	**87.57**	**86.75**
Mean BR1–BR7	88.5	90.1	90.4	88.8	90.5	89.7	**94.2**	**93.7**
Mean BR1–BR15	—	88.6	89.0	—	—	86.7	**91.9**	**91.2**
Mean time BR1–BR7	—	316	183	600	210	34	320	52
Mean time BR1–BR15	—	316	183	—	—	69	320	54
CPU frequency (MHz)	—	400	4×400	200	1,700	2,400	2,600	800

Note. Computing times are in seconds, best results appear in bold, and there is no comparison between CLTRS runs.

For the second CLTRS run (800 MHz PC, set B), the computational effort is much lower than the effort of the calculations with the GRASP algorithm (if cycle frequencies and computing times are compared). Nevertheless, CLTRS achieves better mean volume utilizations for all 15 test cases BR1 to BR15 as well, and the mean improvement still amounts to 1.4% points.

If full support of boxes is required, then CLTRS again performs best for all 15 test cases BR1 to BR15. The parallel GA from Gehring and Bortfeldt (2002) can be attributed the second rank, and similar statements can be made as in the previous comparison with the GRASP method. Compared to the parallel GA, in the first run (2.6 GHz PC, set A) a mean improvement of 2.9% points over the 15 test cases was achieved, whereas the mean improvement was 2.2% points for the other run (800 MHz, set B). Note that the effort spent with the second CLTRS run is certainly lower than the effort spent for the parallel GA.

All in all, CLTRS performs better than the compared algorithms in both the nonsupport and the full-support cases. Moreover, CLTRS is apparently just as suitable for the 3D-CLP variants SLOPP (weakly heterogeneous case) and SKP (strongly heterogeneous case).

The achieved mean volume utilizations state that for the CLTRS method, compliance with the support constraint (C2) "costs" about two percentage points volume utilization (see Tables 1 and 2). Similar results have already been found for the methods from Bortfeldt et al. (2003) and Mack et al. (2004). Therefore, it may be suspected that even independently of the power of a heuristic method, the support constraint (C2) has a considerable effect on the (achievable) volume utilization. This issue is further discussed in Fanslau and Bortfeldt (2008).

4.2. Evaluation of Core Concepts

The following comments reflect in the first line the CLTRS results that were calculated with the 2.6 GHz PC and parameter set A, although examining the other presented results would lead to similar conclusions.

The test results are evaluated first with regard to the generalized block building approach. Averaged over all 1,600 instances, 1,395 simple blocks, but 4,137 general full-support blocks and 9,081 general nonsupport blocks, were generated per instance. The supply of GFS and GNS blocks is therefore much more extensive in comparison with simple blocks. This pays off. In both the nonsupport and the full-support variants, the best solution is found for about 70% of all instances in stage 2, i.e., using general blocks as well. Without the second stage of the search, the mean volume utilization would worsen by about 0.8% points in the nonsupport variant and by around 1.1% points in the full-support variant.

However, the usefulness of general blocks depends greatly on the heterogeneity of the box sets. In the full-support variant, for the first eight test cases BR0 to BR7, the best solution is achieved for 44% of the instances in stage 1, that is, with simple blocks only; whereas for the following eight test cases BR8 to

BR15, the best solution is determined in stage 1 for just 9% of the instances. Similar figures apply for the nonsupport variant. If the three test cases BR13 to BR15 with the most heterogeneous box sets are taken, the volume utilization is increased through the second search stage by around 2.7% points on average in the packing variant and by around 2.0% points on average in the cutting variant! On the one hand, the general blocks introduced here prove to be a key concept for achieving higher volume utilizations for strongly heterogeneous box sets. On the other hand, the selected two-stage search approach conforms to the different suitability of simple and general blocks for weakly and strongly heterogeneous instances.

Second, the special variant of the heuristic tree search, PCTRS, is to be evaluated. A comparison with the TS method from Bortfeldt et al. (2003) and with the SA/TS hybrid from Mack et al. (2004) lends itself for this purpose (see Table 1). Both methods are based on the traditional block building approach: generated packing plans consist of simple blocks. For the test cases BR1 to BR7, the CLTRS now achieves a mean capacity utilization of 94.7% (nonsupport variant) in the first search stage (simple blocks only). For the TS method and the SA/TS hybrid, respectively, the corresponding values are 92.7% and 93.2%. CLTRS clearly dominates the quoted methods even with a restriction to simple blocks. Therefore, it seems that this dominance is above all a result of the special approach to tree search introduced here.

4.3. Additional Experiments and Results

In Table 3 further results for different CLTRS variants are listed that were calculated by means of the 2.6 GHz PC and with parameter set A. For all 16 BR test cases, the standard deviations of the volume utilization are given in percent. The indices for stability St1 and St2 (compare Bischoff and Ratcliff 1995) are shown for the full-support variant, and the mean box support Med-Supp for the nonsupport variant:

—The index St1 indicates the mean number of boxes that support a box that does not lie on the floor of the container. St1 should be as large as possible.

—The index St2 indicates the percentage of boxes in relation to all placed boxes that are not touched on at least three sides by another object (a box or container). St2 should be as small as possible.

—The mean box support Med-Supp shows the mean share of the bottom area (in percent) that is supported from below for those boxes that do not lie on the container floor.

In addition, mean volume utilizations and standard deviations for all 16 BR test cases are presented that were achieved with the following CLTRS variants:

—Nonsupport and no-orientation constraint (C3); i.e., all restrictions of spatial orientations of boxes are ignored.

—Partial support; i.e., the support rate *min_support* (see §3.5) is set to 70% while orientation constraint (C3) is observed.

At first the stability of volume utilizations is considered. The standard deviations are comparatively low in both the nonsupport and the full-support variant (cf., e.g., Eley 2002). The CLTRS method behaves very stably in this sense. The homogenous instances form an exception that can be easily understood. Because there are only three different box dimensions (at maximum) in the homogeneous case, the container

Table 3 Further Test Results of Method CLTRS for the Bischoff and Ratcliff (1995) Test Cases

Test case	Nonsupport		Full-support			Nonsupport, no orientation constraint		Partial support, *min_support* = 70%	
	Std. dev. (%)	Med-Supp (%)	Std. dev. (%)	St1	St2 (%)	Vol. util. (%)	Std. dev. (%)	Vol. util. (%)	Std. dev. (%)
BR0	6.5	94.91	6.5	1.16	3.30	92.58	4.8	89.84	6.5
BR1	1.9	94.19	2.2	1.37	5.45	96.11	1.1	94.54	2.1
BR2	1.2	92.93	1.5	1.40	8.05	96.17	0.7	94.81	1.5
BR3	0.8	90.91	1.0	1.37	10.78	95.97	0.7	94.81	1.0
BR4	0.7	89.73	1.0	1.39	11.78	95.70	0.7	94.50	1.0
BR5	0.6	88.96	0.8	1.39	13.73	95.48	0.6	94.24	0.7
BR6	0.6	86.37	0.7	1.41	15.76	95.20	0.6	93.94	0.8
BR7	0.6	82.21	0.7	1.36	19.50	94.71	0.5	93.31	0.7
BR8	0.5	79.96	0.8	1.38	24.76	94.19	0.5	92.48	0.7
BR9	0.5	76.88	0.7	1.38	30.17	93.80	0.4	91.72	0.6
BR10	0.5	76.56	0.7	1.36	33.21	93.52	0.4	91.16	0.8
BR11	0.5	76.01	0.8	1.33	38.17	93.31	0.4	90.41	0.9
BR12	0.5	74.37	0.7	1.32	40.83	93.01	0.4	90.05	0.7
BR13	0.5	74.07	0.9	1.30	42.89	92.83	0.4	89.61	0.8
BR14	0.5	73.59	0.8	1.27	46.67	92.78	0.4	88.86	0.8
BR15	0.5	59.71	0.8	1.25	50.45	92.61	0.5	88.42	0.9
Mean	1.1	—	1.3	1.3	24.7	94.2	0.8	92.0	1.3

dimensions often cannot be represented to a sufficient extent by combinations of box dimensions. In this case large volume losses will result. For instance, 23 of test case BR0, the globally optimal volume utilization is for example only 65.53%, as a manual calculation proves. As high utilizations are achieved for other homogeneous instances, a relatively large standard deviation is spotted for test case BR0.

The next issue to be dealt with is the loading stability. The full-support variant is examined first. With the block building approach, particularly good values are naturally not achieved for stability criterion St1. This is not changed by the use of general blocks, because these are put together from simple blocks. However, with regard to stability criterion St2, CLTRS achieves much better values for the test cases BR1 to BR7 than many other methods, among which is the heuristic from Bischoff et al. (1995) (cf. Eley 2002).

For the nonsupport variant, the mean box support (Med-Supp in Table 3) achieves high values above 90% for weakly heterogeneous test cases and retains a level above 70% for all test cases except BR15. Thus the nonsupport variant could also be suitable for packing scenarios if the stability requirements are less strict.

The results for the CLTRS variant "Nonsupport, no orientation constraint" show a limited influence of the orientation constraint on the volume utilizations (see Table 1). Averaged over all 16 test cases an improvement of 0.6% points compared to the nonsupport variant results if the orientation constraint (C3) is ignored. For the weakly heterogeneous test cases, the observed improvement of volume utilization is significantly higher and reaches 2.6% points for BR0 and at least 1.0% points for BR1. For the three test cases BR1 to BR3, a mean utilization of 96% or more is achieved.

If partial support ($min\_supp = 70\%$) is required instead of full support, the volume utilizations are only slightly improved (see Table 2). The mean improvement over all 16 test cases amounts to 0.2% points. A significant increase occurs only for strongly heterogeneous test cases; for BR13 to BR15 a mean improvement of the utilization of approximately 0.6% points is observed. Although these results may be disappointing at first glance, they correspond to the following facts. The generation of simple blocks was not changed at all in the partial-support variant, and with the weakly heterogeneous test cases, best solutions very often consist of simple blocks. Although the number of general blocks is more than doubled on average if the support constraint is dropped completely (nonsupport case), this number rises only by (about) 20% if the full-support constraint is replaced by the used partial-support requirement ($min\_support = 70\%$). Finally, the management of residual spaces has not yet been adapted to the partial-support case (this being an issue for further research).

Table 4 Results of Parameter Sensitivity Study with 48 BR Instances

Parameter	Set B	Set C	Set D	Set E	Set F	Set G	Set H
max_bl	10,000	15,000	5,000	10,000	10,000	10,000	10,000
min_fr (%)	98	98	98	99	95	98	98
min_ns	3	3	3	3	3	2	5
time_limit_1	10	10	10	10	10	10	10
time_limit_2	30	30	30	30	30	30	30
Mean volume utilization (%)	92.6	92.4	92.8	93.0	92.3	92.6	92.8

In Table 4 the results of a parameter sensitivity study carried out with the nonsupport variant of CLTRS on the 800 MHz PC are presented. Seven parameter sets were used alternately to calculate a set of 48 BR instances comprising the first three instances of all 16 test cases. In all sets—among them, set B introduced above—the time limits are always set to 10 seconds (stage 1) and 30 seconds (stage 2). The other parameters—namely, maximal number of blocks (*max_bl*), minimal space utilization of blocks (*min_fr*), and minimal search width (*min_ns*)—are varied as follows. In each of the sets C to H one of the three parameters always takes a higher or a lower value compared to set B as indicated in the table.

The calculated mean utilizations show relatively small differences. Other calculations with larger samples also led to the conclusion that there is only a limited influence of the parameters (if taken from reasonable intervals) on the solution quality.

5. Summary

This paper presents the tree search method CLTRS for the three-dimensional container loading problem. Only the CLTRS full-support variant guarantees the full support from below of all packed boxes, whereas the nonsupport variant and the full-support variant observe the guillotine cut constraint.

The CLTRS method is based above all on two concepts. On the one hand, the traditional block building approach is generalized so that blocks with small gaps are now admissible as well as structural elements of packing plans. On the other hand, a special form of tree search, called here PCTRS, is used to pack blocks. The PCTRS makes the search efficient and diverse in that it generates solutions economically and at the same time ensures both the required width and the necessary foresight of the tree search.

On the test of the 1,600 3D-CLP instances from Bischoff and Ratcliff (1995), CLTRS achieves the best volume utilizations up to now for both weakly and strongly heterogeneous instances with acceptable computing times and proves to be suitable for the

problem types SLOPP and SKP. Results were presented as well for instances with homogeneous stocks of boxes, i.e., for the up-to-now rather neglected problem type 3D identical item-packing problem.

In the future, additional constraints (e.g., weight distribution) are to be implemented in CLTRS, and two-dimensional a variant of the algorithm is also planned.

Acknowledgments

The authors thank the three anonymous referees for their valuable comments that greatly helped to improve the presentation of the paper. Special thanks is due to Markus Bremshey for his technical support.

References

Bischoff, E. E. 2004. Three-dimensional packing of items with limited load bearing strength. *Eur. J. Oper. Res.* **168**(3) 952–966.

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

Bischoff, E. E., F. Janetz, M. S. W. Ratcliff. 1995. Loading pallets with non-identical items. *Eur. J. Oper. Res.* **84**(3) 681–692.

Bortfeldt, A., H. Gehring. 2001. A hybrid genetic algorithm for the container loading problem. *Eur. J. Oper. Res.* **131**(1) 143–161.

Bortfeldt, A., H. Gehring, D. Mack. 2003. A parallel tabu search algorithm for solving the container loading problem. *Parallel Comput.* **29**(5) 641–662.

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

Dyckhoff, H., U. Finke. 1992. *Cutting and Packing in Production and Distribution*. Physica-Verlag, Heidelberg, Germany.

Eley, M. 2002. Solving container loading problems by block arrangements. *Eur. J. Oper. Res.* **141**(2) 393–409.

Fanslau, T., A. Bortfeldt. 2008. A tree search algorithm for solving the container loading problem. Diskussionsbeitrag 426. Technical report, Fakultät für Wirtschaftswissenschaft, FernUniversität Hagen, Germany. http://www.fernuni-hagen.de/WINF/menuefrm/publik.htm.

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

Gehring, H., A. Bortfeldt. 1997. A genetic algorithm for solving the container loading problem. *Internat. Trans. Oper. Res.* **4**(4-5) 401–418.

Gehring, H., A. Bortfeldt. 2002. A parallel genetic algorithm for solving the container loading problem. *Internat. Trans. Oper. Res.* **9**(4) 497–511.

Hemminki, J. 1994. Container loading with variable strategies in each layer. Presentation, ESI-X, July 2–15, EURO Summer Institute, Jouy-en-Josas, France.

Hifi, M. 2002. Approximate algorithms for the container loading problem. *Internat. Trans. Oper. Res.* **9**(6) 747–774.

Lim, A., B. Rodrigues, Y. Wang. 2003. A multi-faced buildup algorithm for three-dimensional packing problems. *Omega* **31**(6) 471–481.

Mack, D., A. Bortfeldt, H. Gehring. 2004. A parallel hybrid local search algorithm for the container loading problem. *Internat. Trans. Oper. Res.* **11**(5) 511–533.

Martello, S., D. Pisinger, D. Vigo. 2000. The three-dimensional bin packing problem. *Oper. Res.* **48**(2) 256–267.

Morabito, R., M. Arenales. 1994. An AND/OR-graph approach to the container loading problem. *Internat. Trans. Oper. Res.* **1**(1) 59–73.

Moura, A., J. F. Oliveira. 2005. A GRASP approach to the container-loading problem. *IEEE Intelligent Systems* **20**(4) 50–57.

Parreño, F., R. Alvarez-Valdes, J. F. Oliveira, J. M. Tamarit. 2007. A maximal-space algorithm for the container loading problem. *INFORMS J. Comput.* **20**(3) 412–422.

Pisinger, D. 2002. Heuristics for the container loading problem. *Eur. J. Oper. Res.* **141**(2) 143–153.

Sixt, M. 1996. *Dreidimensionale Packprobleme: Lösungsverfahren basierend auf den Meta-Heuristiken Simulated Annealing und Tabu-Suche*. Peter Lang, Europäischer Verlag der Wissenschaften, Frankfurt am Main, Germany.

Terno, J., G. Scheithauer, U. Sommerweiß, J. Rieme. 2000. An efficient approach for the multi-pallet loading problem. *Eur. J. Oper. Res.* **123**(2) 372–381.

Wäscher, G., H. Haußner, H. Schumann. 2007. An improved typology of cutting and packing problems. *Eur. J. Oper. Res.* **183**(3) 1109–1130.