European Journal of Operational Research 131 (2001) 143–161

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

Theory and Methodology

A hybrid genetic algorithm for the container loading problem

Andreas Bortfeldt *, Hermann Gehring

FernUniversität Hagen, Gesamthochschule – in Hagen, Profilstrasse 8, 58084 Hagen, Germany

Received 7 April 1999; accepted 13 January 2000

Abstract

This paper presents a hybrid genetic algorithm (GA) for the container loading problem with boxes of different sizes and a single container for loading. Generated stowage plans include several vertical layers each containing several boxes. Within the procedure, stowage plans are represented by complex data structures closely related to the problem. To generate offspring, specific genetic operators are used that are based on an integrated greedy heuristic. The process takes several practical constraints into account. Extensive test calculations including procedures from other authors vouch for the good performance of the GA, above all for problems with strongly heterogeneous boxes. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Packing; Container loading problem; Three-dimensional knapsack problem; Genetic algorithm

1. Introduction and formulation of the problem

The efficient stowage of goods in means of transport can often be modelled as a container loading problem. In this problem, a set of rectangular packages (boxes) has to be arranged in one or more rectangular containers in such a way that optimum use is made of the available container space. Where necessary, additional constraints must be met, e.g., a weight constraint for the stowed load.

In the literature, container loading problems are differentiated in several ways (cf. Dyckhoff and Finke, 1992). Firstly, problems in which the complete load has to be stowed are differentiated from those which tolerate some goods to be left behind. Problems of the first type are known in the literature as (three-dimensional) bin-packing problems, and problems of the second type as (three-dimensional) knapsack problems. While bin-packing problems aim, e.g., at minimising the required container costs, the target of knapsack problems is usually to maximize the stowed volume or the unit contribution of freight.

Another type of differentiation presupposes the definition of a type of box, which is agreed as follows. Two boxes are exactly the same type if, given a suitable spatial orientation, they coincide in all three lateral dimensions. This differentiates between problems with a homogeneous set of boxes (one type of box only), a weakly heterogeneous set

* Corresponding author. Tel.: +49-2331-9874431.

E-mail address: andreas.bortfeldt@fernuni-hagen.de (A. Bortfeldt).

of boxes (few box types, many boxes of each type), and a strongly heterogeneous set of boxes (many box types, few boxes of each type).

The subject of the paper is the container loading problem of the knapsack type with a single container, whereby the emphasis is on the case of a strongly heterogeneous set of boxes. Further, various practical constraints will be taken into account. The problem under consideration may be stated as follows: a given container is to be loaded with a subset of a given set of boxes in such a way that all boxes are positioned in a feasible way, the stowed volume is maximised and the constraints are met. A box is considered to be stowed in a feasible way if it is arranged parallel to the side walls of the container, lies completely within the container and does not overlap with another box.

The following selection of constraints taken from the great number of constraints found in practice (cf. e.g. Bischoff and Ratcliff, 1995) will be included in the problem. Further, it is assumed that the container lies in the first octant of a three-dimensional coordinate system with a bottom corner in the coordinate origin, and the depth, width and height of the container are oriented in accordance with the directions of the x-, y- and z-axes, respectively. It is also assumed that the centre of gravity of each box coincides with its geometric centre.

(C1) *Orientation constraint*: Each box can be arranged originally in the container in a maximum of 6 "rotation variants". For each box, up to 5 rotation variants may be prohibited by means of an orientation constraint. For example, a side dimension may not be used as the height, if this is required by the position of the items in the box.

(C2) *Stability constraint*: In the interests of high stability for the load it is demanded that the bottom area of all boxes not placed directly on the container floor is completely supported by the upper surface of other boxes.

(C3) *Stacking constraint*: Boxes may not be placed on top of certain other boxes, e.g., because the contents must not be subjected to vertical loads.

(C4) *Weight constraint*: The total weight of the stowed boxes may not exceed a given upper limit, because, e.g., the container is not designed for a higher net weight of the load.

(C5) *Balance constraint*: The distance between the x-coordinates of the centre of gravity of the arranged boxes and the middle of the container depth may not exceed a given percentage of the container depth. There is an analogue condition for the y-coordinates of the arranged boxes.

Problem-specific heuristics for container loading problems with a single container and a weakly heterogeneous set of boxes have been suggested in the recent past by Loh and Nee (1992), Ngoi et al. (1994), Bischoff et al. (1995) and Bischoff and Ratcliff (1995). Heuristic procedures for single container problems tailored to a strongly heterogeneous set of boxes were introduced by Gehring et al. (1990), Portmann (1990) and Scheithauer (1992).

More general meta-heuristic search concepts, such as genetic algorithms (GAs), tabu search or informed graph search, were applied to different types of container loading problems and pallet loading problems, respectively, in papers by Prosser (1988), Lin et al. (1993), Bortfeldt (1994), Morabito and Arenales (1994), Sixt (1996), Gehring and Bortfeldt (1997) and Bortfeldt and Gehring (1998).

Although some efficient procedures for three-dimensional packing problems already exist, the development of methods able to achieve high or near optimal volume utilisations remains a challenge. Moreover, deficiencies of the proposed approaches consist, from a practical point of view, in the lack of consideration of constraints. Hence, in this paper, a hybrid GA is proposed which considers the constraints specified above. On the one hand, a suitable hybridisation of GAs is often the key to success (cf. e.g. Reeves, 1993a,b; Gehring and Schütz, 1994; Kopfer et al., 1994). On the other hand, constraints may easily be integrated in the genetic search concept.

As the concise overview from Reeves (1993b) makes clear, hybridisation of GAs can be carried out in several ways. However, their "common denominator" is always formed by the enrichment of a GA with additional problem-specific knowledge.

A consistent method of hybridisation has already been recommended by Davis (1991) and systemised by Michalewicz (1992). The core of this approach, which Michalewicz summarised in the equation "GAs + data structures = evolution programs", may be outlined as follows:

- Solutions are represented naturally through more complex data structures such as graphs or matrices, i.e., solutions are not coded by means of strings via an alphabet of less cardinality, as is usual for GAs.
- Accordingly, the operators used to generate the offspring are designed specifically for the problem, i.e., these operators must be suitable for manipulating the complex data structures which represent the solutions.

A characteristic of this approach is the use of a problem representation close to the phenotype level. Thus, typical difficulties are avoided occurring with the design of GAs, such as the occurrence of inadmissible or redundant chromosomes. In addition, the procedural effort is displaced from decoding the chromosomes to applying the operators, which are usually much more complex. As a promising approach the hybridisation method of Michalewicz (1992) is included in the proposed algorithm.

The remainder of this paper is structured as follows: Section 2 describes the overall procedural solution concept as well as the underlying problem representation. Section 3 specifies a subordinate heuristic, called basic procedure, which is used to generate single stowage plans (individuals), whereas the manipulation of populations of individuals by means of a hybrid GA is specified in Section 4. After the consideration of the mentioned constraints is explained in Section 5, an extensive test of the procedure is documented in Section 6. Finally, a summary of the paper is given in Section 7.

2. Solution concept

The overall procedural solution concept can be formulated as follows:

- The hybrid GA is used to generate stowage plans with a layer-type structure. Each generated solution consists of several vertical, cuboid-shaped and non-overlapping layers parallel to the face walls of the container. The layers follow one another without any gaps. Each layer contains one or more boxes which do not project into adjacent layers.
- The height and width of a layer coincide with the corresponding container dimensions. The depth of the layer results from the depth dimension of the layer-defining box. Fig. 1 shows the layer-type structure of generated stowage plans.
- Stowage plans are generated by means of an integrated subordinate heuristic, the basic heuristic:
  1. At the beginning of the procedure, the basic heuristic is used to provide complete stowage plans in population strength as starting individuals for the evolution.
  2. For the generation of subsequent generations problem-specific operators, namely, *crossover* and *mutation*, are used. These transfer some unchanged layers of the parent individuals to the offspring. Since an offspring still does not generally represent a complete solution, additional layers are generated by means of the basic heuristic.
  3. The completion of partial solutions is performed in different ways which are caused by modifications of the *crossover* and *mutation* operators. Hence, the basic heuristic is used in a total of three variants. Each of the stowage plan layers generated in the course of the procedure is generated by means of the basic heuristic.

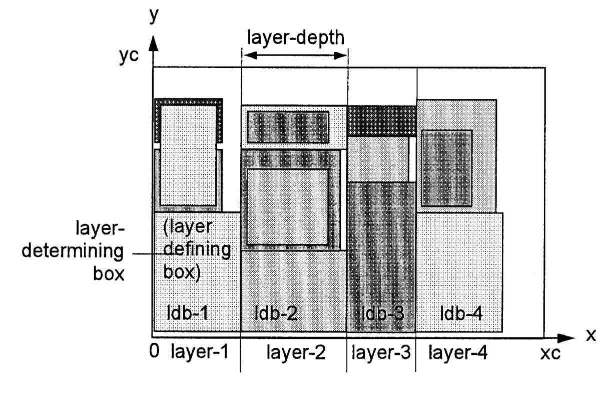


Fig. 1. Stowage plan with 4 layers (overhead view).

- The basic heuristic and the genetic operators are designed so that the sequence of layers of stowage plans is initially of no concern. For this reason, stowage plans are represented internally in the procedure through a data structure which does not yet specify the sequence of the layers. The sequence of the layers in the container is determined in a final step.

The outlined solution concept is based on a problem representation which includes the following data structures and identifiers:

1. *Container*: While *xc*, *yc* and *zc* denote the depth, width and height of a container, the container volume is denoted by *vc*.

2. *Boxes*: The set of boxes is kept in a vector *BList*(*b*), $b = 1, \ldots, nb$, where *nb* denotes the number of all boxes. A vector element *BList*(*b*) or a single box is defined by its three side dimensions and the box volume. The vector *BList* is sorted in descending order in accordance with the box volume. Subsets of the set of boxes are given through the corresponding index sets. Finally, *BAll* denotes the set of all boxes: $BAll = \{1, \ldots, nb\}$.

3. *Rotation variants*: The 6 maximum possible spatial rotation variants of a box placed in the container are indexed from 1 to 6 in a suitable way.

4. *Box placing*: An individual box placing is given by a structure *pl* with the following components:
- box identifier *b*;
- coordinates of the reference corner of the placed box *ox*, *oy* and *oz*;
- rotation variant of the box *rv*.

The reference corner of a stowed box is the left-hand bottom rear corner. The *y*- and *z*-coordinates are given absolutely. The *x*-coordinates are related to the rear wall of the appropriate layer.

5. *Layer*: A single filled layer is represented by a structure *l* including the following components:
- layer utilisation *vutil*;
- layer depth *d*;
- number of boxes placed in layer *npl*;
- vector of the box placings *Pl*(*i*), $i = 1, \ldots, npl$.

An element of the placing vector *l.Pl*(*i*) corresponds to a single box placing (cf. 4). The layer depth *l.d* is given by the *x*-dimension of the layer-determining box, while layer utilisation *l.vutil* is determined as a quotient of the volume of the placed boxes and the layer volume $l.d \cdot yc \cdot zc$. The operator *B*(*l*) determines the index set of stowed boxes belonging to the placing vector of a layer *l*.

6. *Stowage plan*: A stowage plan is represented by a structure *s* composed of the following components:
- packed box volume *v*;
- number of generated layers *nl*;
- set of generated layers *L*;
- (still) remaining free container depth *xcfree*;
- set of the (still) free, i.e., unstowed, boxes *Bfree*.

The layers of a stowage plan are represented by an unsorted set structure. The elements of this set are layers in the representation agreed under 5. If the stowage plan *s* is still incomplete the free container depth *s.xcfree* and the set of unstowed boxes *s.Bfree* define completely the remaining rest problem.

3. The basic heuristic

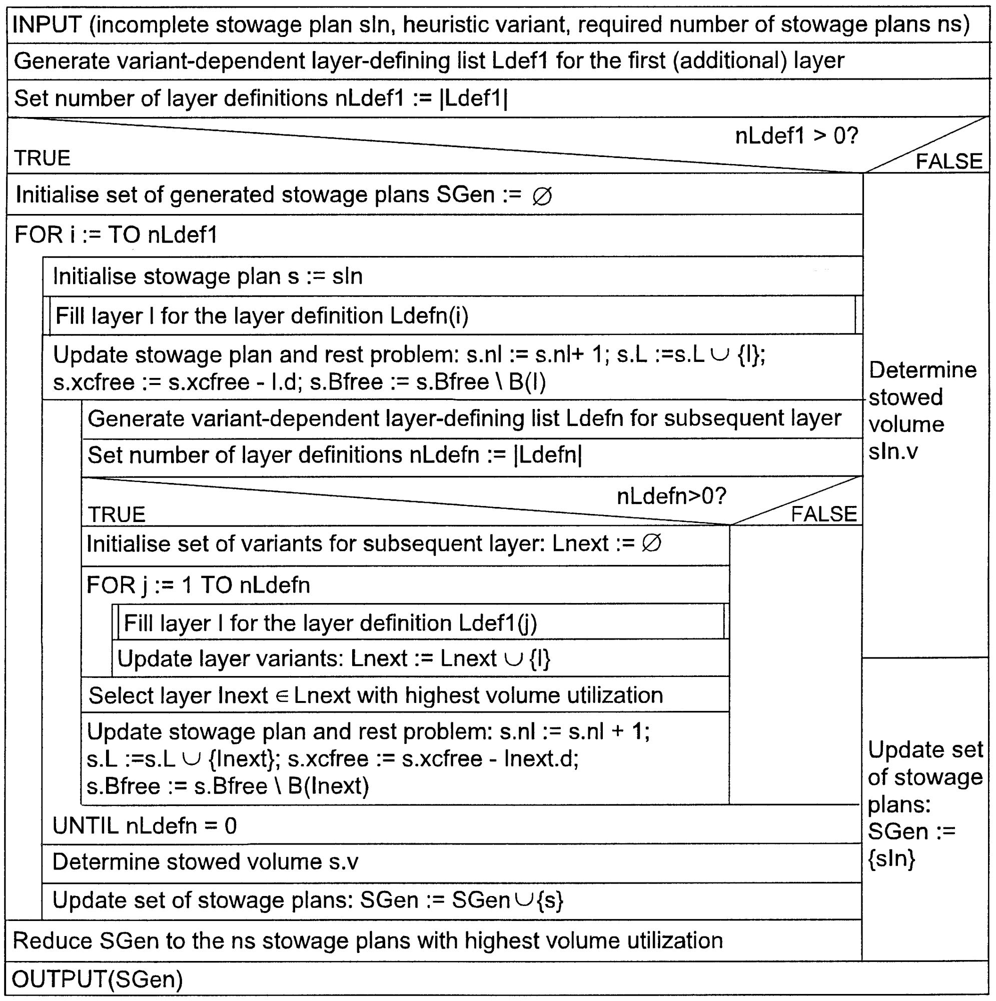
The basic heuristic used to generate and complete stowage plans is derived from the heuristic developed by Gehring et al. (1990). While the general procedure of this heuristic is retained, the heuristic elements and rules are largely replaced.

Layers are generated in two steps:
- A layer is initially defined by stipulating a layer-determining box and its rotation variant.
- The layer is then filled with the layer-determining box and in general with other boxes. The filling is performed with the core procedure of the basic heuristic, denoted by *fill-layer*.

In the following sections, the rough algorithm of the basic heuristic and the core procedure *fill-layer* are described.

3.1. Rough algorithm and variants of the basic heuristic

The procedure of the basic heuristic (cf. Fig. 2) is initially characterised independently of its different variants:
- At the *start* a still incomplete stowage plan *sIn* is passed to the basic heuristic which contains in particular the data of the rest problem.



```
INPUT (incomplete stowage plan sIn, heuristic variant, required number of stowage plans ns)
Generate variant-dependent layer-defining list Ldef1 for the first (additional) layer
Set number of layer definitions nLdef1 := |Ldef1|
nLdef1 > 0?
TRUE
    Initialise set of generated stowage plans SGen := ∅
    FOR i := TO nLdef1
        Initialise stowage plan s := sIn
        Fill layer l for the layer definition Ldefn(i)
        Update stowage plan and rest problem: s.nl := s.nl+ 1; s.L :=s.L ∪ {l};
        s.xcfree := s.xcfree - l.d; s.Bfree := s.Bfree \ B(l)
            Generate variant-dependent layer-defining list Ldefn for subsequent layer
            Set number of layer definitions nLdefn := |Ldefn|
            nLdefn>0?
            TRUE
                Initialise set of variants for subsequent layer: Lnext := ∅
                FOR j := 1 TO nLdefn
                    Fill layer l for the layer definition Ldef1(j)
                    Update layer variants: Lnext := Lnext ∪ {l}
                Select layer lnext ∈ Lnext with highest volume utilization
                Update stowage plan and rest problem: s.nl := s.nl + 1;
                s.L :=s.L ∪ {lnext}; s.xcfree := s.xcfree - lnext.d;
                s.Bfree := s.Bfree \ B(lnext)
            FALSE
        UNTIL nLdefn = 0
        Determine stowed volume s.v
        Update set of stowage plans: SGen := SGen ∪ {s}
    Reduce SGen to the ns stowage plans with highest volume utilization
FALSE
    Determine stowed volume sIn.v
    Update set of stowage plans: SGen := {sIn}
OUTPUT(SGen)
```

Fig. 2. Rough algorithm of the basic heuristic.

- First, a list of feasible layer definitions is generated. A layer definition includes the data of a layer-determining box *ldb* and its rotation variant *rvldb*. A layer definition is feasible if the box *ldb* is still free and can be placed in the remaining container area with its rotation variant *rvldb*.
- For each derived layer definition $Ldef1(i)$, $i = 1, \ldots, nLdef1$, of the first new layer a complete stowage plan is generated. This plan is initialised with the passed incomplete stowage plan *sIn* and then extended with a layer generated in accordance with the current layer definition $Ldef1(i)$. On the extension of the second, third, etc., new layers, a layer is generated experimentally for each layer definition of the prior determined list *Ldefn* of feasible layer definitions; the layer with the highest volume layer utilisation is added to the stowage plan.
- The generation of a stowage plan terminates if the layer definition list of the next layer is empty. In this case, the total packed box volume is computed as the objective function value and the stowage plan is inserted into set *SGen*. Before the generated stowage plan set *SGen* is put out, it may be reduced to the *ns* best stowage

plans in accordance with the required number of stowage plans.

The three variants of the basic procedure are designated *start*, *crossover* and *mutation*. The differences between the three variants are as follows:

- In the case of the variant *start*, an empty stowage plan is passed whose rest problem is equal to the starting problem. In the two other variants, the passed incomplete stowage plan contains at least one layer.
- With the variant *start* stowage plans are generated in accordance with the given population strength of the GA. In the two other variants, the basic heuristic shall only generate one complete stowage plan in each case. If, however, several complete stowage plans were derived from the passed incomplete solution, only the best is returned.
- Layer definition lists are generated as follows:
  - Each layer definition list is initially generated to the full extent and sorted in accordance with the box volume of the layer-determining boxes. In addition, layer definitions on the same layer-determining box are sorted in descending order in accordance with the x-dimension and subordinately in accordance with the z-dimension of the layer-determining box.
  - In the case of the variant *start*, the list of the first layer is retained completely, while the lists for the following layers are reduced to the first element.
  - In the case of the variant *crossover*, only the first $qldb_1$% of all layer definitions are left for the first new layer, while the first $qldb_2$% of all layer definitions remain for the following layers.
  - In the case of the variant *mutation*, only a single layer definition among the first $qldb_3$% of the original existing definitions is selected at random for the lists of the new layers.

The generation of layer definition lists follows the principle that larger volume boxes should be preferred when stowing. The parameter $qldb_3$ for the heuristic variant *mutation* merely defines a selection range for layer definitions. On the other hand, the parameters $qldb_1$ and $qldb_2$ serve the flexible control of the trade-off between the expected solution quality and the effort for the particularly "expensive" variant *crossover*.

3.2. Filling a layer

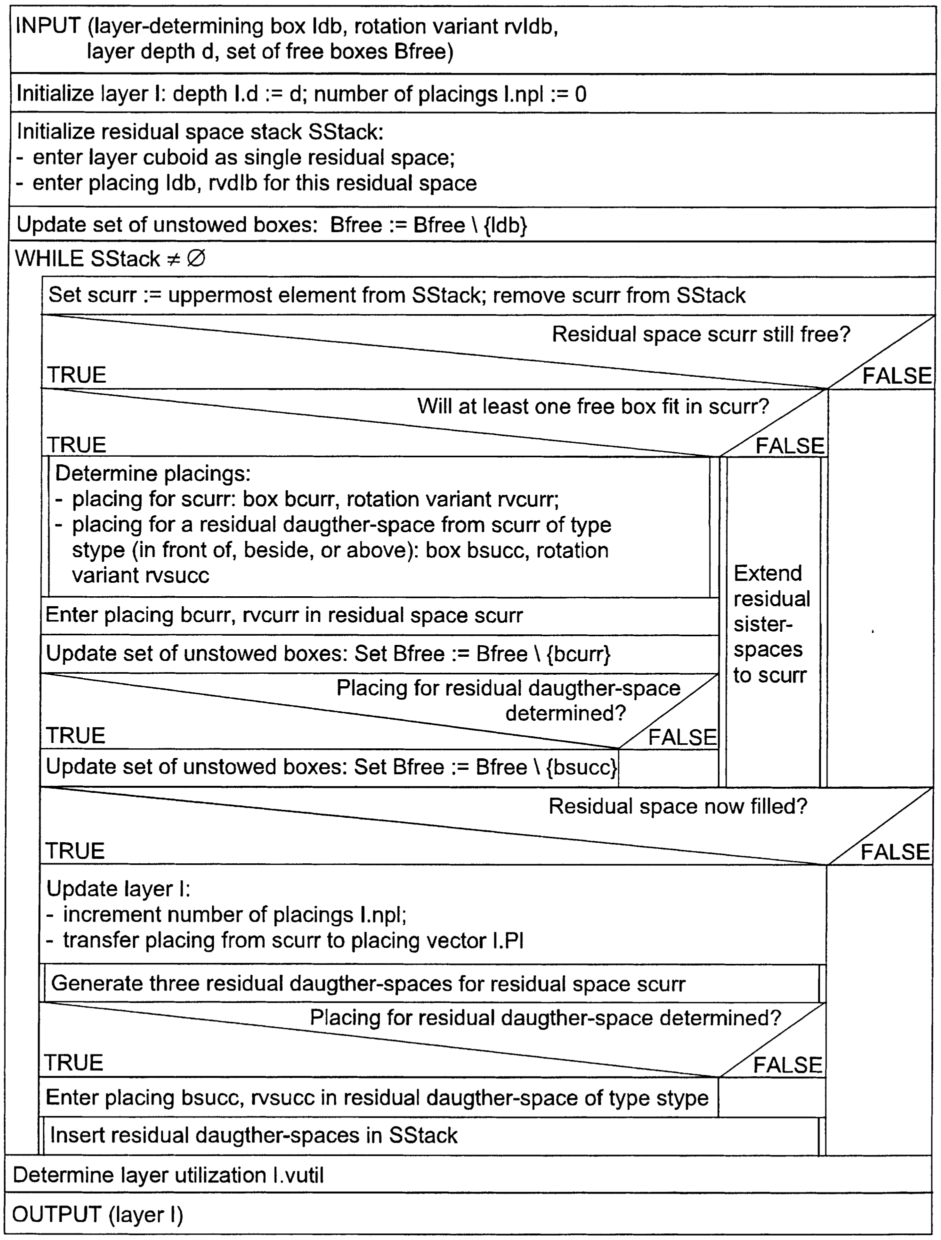
A defined layer is filled by means of the procedure *fill-layer* (cf. Fig. 3), which arranges the layer-determining box with the given rotation variant and – in general – other boxes in the layer cuboid.

In order to fill a layer, residual spaces, i.e., free rectangular spaces within the layer, are filled step by step. In each step, a box is placed in the reference corner of a residual space. The reference corner is the bottom rear left-hand corner of the residual space. The filling of a residual space leads to three additional residual daughter-spaces, within this space, namely, *in front of*, *beside* and *above* the placed box. These are collected and processed later. The filling of a layer terminates, if the set of fillable residual spaces is empty.

The set of (fillable) residual spaces is kept in a stack. This guarantees that the later generated – and therefore smaller – residual spaces are processed earlier, which enhances the probability that these can be filled. An element of the stack is initially characterised by its side dimensions and the coordinates of its reference corner. If a box is placed in the residual space, its index and the applied rotation variant are added.

While processing a residual space 4 distinguishable cases may occur:

1. The residual space is still empty and can be filled with at least one box. In this case, two box placings are usually determined simultaneously. The first is determined for the current residual space and is entered in this immediately. The second placing is destined for a residual daughter-space *in front of*, *beside* or *above* the box placed in the current residual space and can only be entered after the generation of this residual space. Both placed boxes are removed from the set of free boxes. However, only the first placing is taken over in the layer.
2. In deviation from case 1, only one placing is determined for a residual space. Hence, all three and still empty residual daughter-spaces are inserted into the stack.



```
INPUT (layer-determining box ldb, rotation variant rvldb,
       layer depth d, set of free boxes Bfree)
Initialize layer l: depth l.d := d; number of placings l.npl := 0
Initialize residual space stack SStack:
- enter layer cuboid as single residual space;
- enter placing ldb, rvdlb for this residual space
Update set of unstowed boxes: Bfree := Bfree \ {ldb}
WHILE SStack ≠ ∅
    Set scurr := uppermost element from SStack; remove scurr from SStack
    Residual space scurr still free?
    TRUE
        Will at least one free box fit in scurr?
        TRUE
            Determine placings:
            - placing for scurr: box bcurr, rotation variant rvcurr;
            - placing for a residual daugther-space from scurr of type
              stype (in front of, beside, or above): box bsucc, rotation
              variant rvsucc
            Enter placing bcurr, rvcurr in residual space scurr
            Update set of unstowed boxes: Set Bfree := Bfree \ {bcurr}
            Placing for residual daugther-space
                                   determined?
            TRUE
                Update set of unstowed boxes: Set Bfree := Bfree \ {bsucc}
            FALSE
        FALSE
            Extend
            residual
            sister-
            spaces
            to scurr
    FALSE
    Residual space now filled?
    TRUE
        Update layer l:
        - increment number of placings l.npl;
        - transfer placing from scurr to placing vector l.Pl
        Generate three residual daugther-spaces for residual space scurr
        Placing for residual daugther-space determined?
        TRUE
            Enter placing bsucc, rvsucc in residual daugther-space of type stype
        FALSE
        Insert residual daugther-spaces in SStack
    FALSE
Determine layer utilization l.vutil
OUTPUT (layer l)
```

Fig. 3. Algorithm of the procedure *fill-layer*.

3. The residual space removed from the stack is already filled. This case occurs, if the residual space was already loaded as the daughter of a previously processed residual space. Moreover, this case occurs also for the first residual space to be processed (cf. Fig. 3). The placing in the residual space is now taken over in the layer. All residual daughter-spaces are inserted into the stack as empty spaces.
4. An empty residual space cannot be loaded with any of the free boxes. In this case, an attempt is made to transfer at least parts of the residual

space to residual sister-spaces, i.e., residual spaces which were generated together with the non-fillable residual space. Residual daughter-spaces are not generated.

The double lined steps in Fig. 3 require further explanation. These steps are the determination of placings for a residual space, the generation and insertion of residual daughter-spaces in the residual space stack, and the extension of residual sister-spaces.

3.2.1. Determining placings for a residual space

The determination of at maximum two placings for a residual space involves:

1. selecting one or two boxes to be placed;
2. stipulating the residual daughter-space for the second box;
3. determining the rotation variant for the selected boxes;
4. allocating the selected boxes to the current and the residual daughter-space.

Decisions 1–4 are made successively on the basis of the following rules. In addition, a placement of two boxes is excluded if one box lies on the top of the other in a way that lateral overhang occurs. If a single box is selected in decision 1, then its rotation variant has to be determined (cf. rule (R3)).

(R1) Rule for selecting one or two boxes to be allocated:

Rule (R1) considers the set of all pairs of free boxes with the property that both boxes can be arranged completely in the residual space *beside*, *in front of* or *above* one another. Included are also degenerate box pairs consisting of a single box only. The box pair with the largest overall volume is selected from the set for allocation.

(R2) Rule for stipulating the residual daughter-space for the second box:

In general, there exist three positioning variants for the box pair selected with rule (R1): *beside*, *in front of* and *above* (cf. Fig. 4). The smaller the residual space dimension characterising a variant is the higher is its priority. Let, e.g., the x-dimension of the residual space be smaller than the y-dimension. Then the positioning variant *in front of* has higher priority than the variant *beside*. According to the variant with the highest priority the residual daughter-space for the second box is determined. If, e.g., the position with the highest priority is the position *in front of*, then the residual daughter-space of the second box is of the type *in front of*.

(R3) Rules for determining the rotation variants of the selected boxes:

For determining the rotation variants of the selected boxes two rules, designated (R3.1) and (R3.2), are used alternatively within the GA (cf. Section 4).

By means of rule (R3.1) the rotation variants of the two boxes are stipulated in such a way that the x-dimension of each box, and, subordinately, the z-dimension of each box, is as small as possible. This rule is applied at first to the box with the larger volume. In a second step, the rule is then used to determine the rotation variant of the other box.

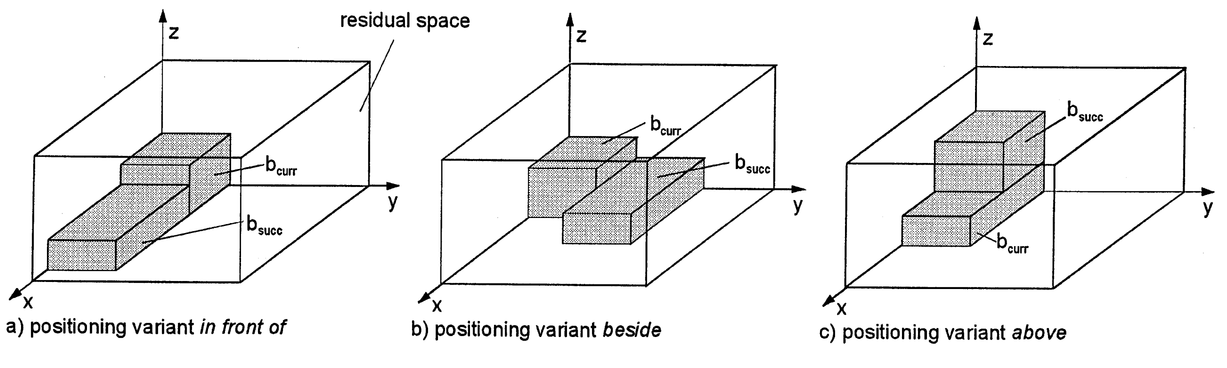

residual space
z
y
x
b_curr
b_succ
a) positioning variant *in front of*
b) positioning variant *beside*
c) positioning variant *above*

Fig. 4. Positioning variations for two boxes in a single residual space.

Rule (R3.2) serves the selection of that feasible combination of two rotation variants for which the sum of the box dimensions in the z-direction and, subordinately, the sum of the dimensions in the x-direction, are as large as possible.

If only a single box has been determined with rule (R1), rule (R3.1) or rule (R3.2) is to be applied analogously.

(R4) Rule for allocating boxes to the current and the residual daughter-space:

If one of the positioning variants *in front of* or *beside* and the corresponding residual daughter-space were determined using rule (R2), then the box with the largest z-dimension is placed in the reference corner of the current residual space. The other box is placed accordingly either *in front of* or *beside* the first box.

If the residual daughter-space type *above* was determined for the second box, then the box with the largest base is placed in the reference corner of the current residual space. The other box is placed immediately on top of the first box.

3.2.2. Generating residual daughter-spaces

For a residual space s_{curr}, in whose reference corner a box b_{curr} was placed, three residual daughter-spaces – denoted by $s_{infront}$, s_{beside} and s_{above} – are generated as follows:

- The free cuboid lying on the top of b_{curr} and reaching to the upper surface of the subdivided residual space s_{curr} is selected as the residual daughter-space s_{above}.
- The residual spaces $s_{infront}$ and s_{beside} are determined so that they lie on the base of the subdivided residual space s_{curr}, and reach to its upper surface. For this purpose, the L-shaped part of the base remaining after the placing of box b_{curr} is divided into two rectangles. As Fig. 5 shows, there exist two subdivision variants denoted by *in front of large* and *beside large.*
- Initially, the subdivision variant is selected for which the larger of the two resulting residual spaces has the maximum base area. Thus, the chance is enhanced that at least one of the subsidiary residual spaces can be loaded.
- However, if in the case of the selection of subdivision variant *beside large* the placing of box b_{succ} is destined for the residual space $s_{infront}$ and the box b_{succ} would project into the residual space s_{beside}, then the alternative breakdown variant *in front of large* is selected instead. If a placing is planned in the residual space s_{beside}, then analogous considerations apply.

3.2.3. Extending residual sister-spaces

If a residual space proves to be unfillable, then defined rectangular parts of this space should be transferred to those of its residual sister-spaces which have not yet been processed. Thus, the loss of space is kept as low as possible. However, a transfer of partial spaces is intended only between two residual sister-spaces $s_{infront}$ and s_{beside} which are residual daughter-spaces of a residual space s_{curr} filled with a box b_{curr}.

Assume that on the occasion of generating residual spaces for s_{curr} the subdivision variant *in front of large* has been realised and that the residual space $s_{infront}$ is no longer fillable. Let, at the

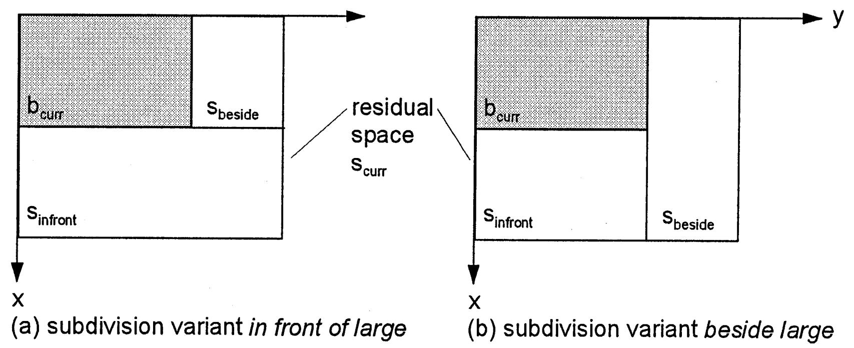

b_curr
s_beside
s_infront
residual space s_curr
b_curr
s_infront
s_beside
y
x
x
(a) subdivision variant in front of large
(b) subdivision variant beside large

Fig. 5. Subdivision variants of a residual space (overhead view).

same time, the residual sister-space s_{beside} still be in the residual space stack. Then the subdivision variant *beside large* is subsequently selected for the residual space s_{curr}. This way a free cuboid diagonally opposite box b_{curr} is transferred from $s_{infront}$ to s_{beside} (cf. Fig. 6). An analogous space transfer may be carried out from a residual space s_{beside} to the residual sister-space $s_{infront}$.

3.2.4. Inserting residual daughter-spaces into the residual space stack

Residual daughter-spaces of a residual space s_{curr} filled with a box b_{curr} are generally inserted into the residual space stack after their generation.

However, a residual daughter-space may be merged with a residual space kept in the stack. This enlarges the latter residual space, though the stack itself does not grow.

A merger of a residual space s_{new} with a residual space s_{old} in the stack is carried out under the following conditions. A box placing for either s_{new} or s_{old} has not yet been determined. In addition, s_{new} and s_{old} must border each other horizontally and have dimensions that enable them together to form a cuboid within the layer.

If new residual spaces cannot be merged with existing spaces, then a new residual space s_{above} is always inserted into the stack as the first residual daughter-space. If the subdivision variant in *front of large* was realized, then the remaining residual spaces are inserted into the stack in the sequence s_{beside} before $s_{infront}$, otherwise they are inserted in the reverse order. The selected sequence guarantees that a residual space that can be used as the donor of a residual space extension, is always processed before a residual sister-space that might be the recipient.

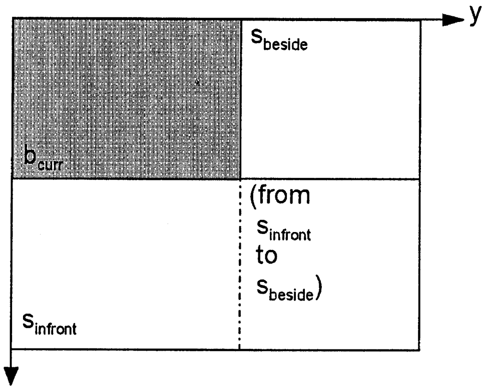


Fig. 6. Extending a residual space s_{beside} by a part of its residual sister-space $s_{infront}$.

4. The genetic algorithm

Since the problem representation used by the GA has already been specified (cf. Section 2), the generic and procedural aspects are now in the foreground. Fig. 7 shows the rough procedure of the GA.

The size of the population *npop* is held constant during evolution. For reproduction the model of generational replacement is used.

Solutions of a subsequent generation are generated by means of a *crossover* operator and a *mutation* operator. The *mutation* operator has two variants, called *standard mutation* and *merger mutation*. In addition, a reproduction operator is used to ensure the survival of the best previous solution.

After the reproduction of the *nrep*, $nrep \geqslant 1$, best solutions of the old generation, the subsequent generation is initially filled to population strength. For this purpose, the *crossover* and the *standard mutation* are applied alternatively with the complementary and constant probabilities p_{cross} and p_{mut}, $p_{cross} + p_{mut} = 1$. Subsequently, a number of *nmerge* additional solutions are generated through the *merger mutation* and inserted in the new generation. At last the new generation is reduced to the *npop* best solutions.

Duplicates are not permitted in either the *start* generation or in any of the subsequent generations. If a generated solution proves to be a duplicate of an individual already present in the respective generation, it is not included in the generation and counts as not generated.

Finally, the sequence of the layers in the container is to be determined for the best generated stowage plan. While this sequence can be chosen arbitrarily for constraint-free problems, the appropriate determination of the layer sequence in the case of a balance constraint (C5) follows a concept presented in Section 5.

```
INPUT (container data, box data, parameters)
Generate npop stowage plans for the start generation using the basic heuristic (variant start)
Evaluate the start generation and use it as the current generation
WHILE termination criterion is not met
    Reproduce the best nrep of the stowage plans of the current generation for the
    subsequent generation
    WHILE subsequent generation contains fewer than npop stowage plans
        Select operator op ∈ {crossover, mutation} randomly with the application probabilities
        pcross and pmut
        op = crossover?
        TRUE
            Select stowage plans p1, p2
            of the current generation
            Generate descendant o from
            p1 and p2 by crossover
        FALSE
            Select stowage plan p
            of the current generation
            Generate descendant o from p
            by standard mutation
    FOR i := 1 TO nmerge
        Select stowage plan p of the current generation
        Generate descendant o from p through merger mutation
    Evaluate and reduce the subsequent generation and use it as the current generation
Determine a layer arrangement for the best generated stowage plan sopt
OUTPUT (sopt)
```

Fig. 7. Rough procedure of the GA.

The following detailed specification of the GA covers the fitness evaluation and the selection of solutions, the genetic operators and the configuration of the GA.

4.1. Fitness evaluation and selection

The generation of offspring should consider two aspects: on the one hand, the selection should prefer solutions with high objective function values and on the other hand, premature homogenisation of the population should be prevented. Two selection methods are combined to take account of both aspects:

- the ranking selection with linear normalisation, the advantages of which are generally familiar (cf. e.g. Whitley, 1989; Reeves, 1993a; Falkenauer, 1998);
- the random selection of solutions with equally distributed selection probabilities.

Ranks are determined on the basis of objective function values as follows:

- The solutions s_i, $i = 1, \ldots, npop$, of the current generation are initially sorted in descending order in accordance with their objective function values, i.e., in accordance with the stowed box volume.
- The rank or (relative) fitness $f(s_i)$ of a solution s_i is now defined in accordance with:
    - $f(s_i) = f_{\max} - (i-1) \cdot d,\ i = 1, \ldots, npop;$
    - here, $f_{\max}$ denotes a positive fitness maximum and d a positive fitness decrement, whereby $f_{\max} = 2/(npop+1)$ and $d = 2/(npop \cdot (npop+1))$

Note that the designated fitness values $f(s_i)$, $i = 1, \ldots, npop$, are directly suitable as selection probabilities. The selection concept may now be specified more precisely:

- Both a solution for *mutation* and the first parent solution required for a *crossover* are selected through ranking selection in accordance with the defined fitness evaluation.

- The second parent for the *crossover* is determined by means of random selection.
- When offspring is generated by *crossover*, each parent pair is used only once during a generation. In the event of the selection of a parent pair that has already been used, the selection of the parents must be repeated.

4.2. The crossover operator

The *crossover* operator is defined through the algorithm shown in Fig. 8. According to Fig. 8 a descendant is generated in two phases:

1. *Layer transfer*: In the first phase, layers from both parents are transferred unchanged to the descendant. A parent layer can be transferred to the descendant only if the following conditions are met:

(a) All the boxes placed in the parent layer belong to the set of the free boxes of the descendant *o.Bfree*, i.e., they have not yet been allocated in the offspring.

(b) The parent layer fits into the still free container section *o.xcfree* of the offspring.

Constraint (a) ensures that a parent layer cannot be transferred more than once. If the set *Lnext* of the transferable parent layers is not empty at a defined transfer stage, the layer with maximum utilisation is determined and transferred to the offspring. Otherwise, the layer transfer is terminated.

2. *Layer extension*: Subsequent to the layer transfer, the basic heuristic is applied to the descendant in the variant *crossover*. This generally extends the descendant into a complete stowage plan through additional, newly generated layers.

When the *crossover* operator is applied, it may happen that a parent stowage plan is reproduced identically. If this is the case, the *crossover* is terminated without results and repeated with a newly selected parent pair.

4.3. The mutation operator

The procedure of *standard mutation* is described by the algorithm shown in Fig. 9.

In a *mutation* a descendant is also generated in two phases:

1. *Layer transfer*: In the first phase, the best layers of the parent solution, i.e., the layers with

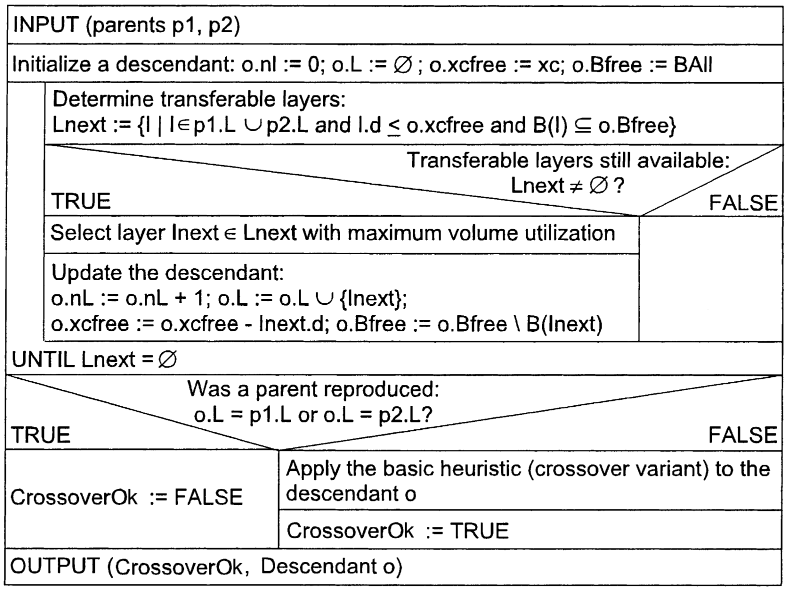

INPUT (parents p1, p2)
Initialize a descendant: o.nl := 0; o.L := ∅ ; o.xcfree := xc; o.Bfree := BAll
Determine transferable layers:
Lnext := {l | l∈p1.L ∪ p2.L and l.d ≤ o.xcfree and B(l) ⊆ o.Bfree}
Transferable layers still available:
Lnext ≠ ∅ ?
TRUE
FALSE
Select layer lnext ∈ Lnext with maximum volume utilization
Update the descendant:
o.nL := o.nL + 1; o.L := o.L ∪ {lnext};
o.xcfree := o.xcfree - lnext.d; o.Bfree := o.Bfree \ B(lnext)
UNTIL Lnext = ∅
Was a parent reproduced:
o.L = p1.L or o.L = p2.L?
TRUE
FALSE
CrossoverOk := FALSE
Apply the basic heuristic (crossover variant) to the descendant o
CrossoverOk := TRUE
OUTPUT (CrossoverOk, Descendant o)

Fig. 8. Algorithm of the crossover operator.

```
INPUT (parent p)
Initialize descendant: o.nl := 0; o.L := ∅ ; o.xcfree := xc; o.Bfree := BAll
Select number of layers to be transferred nlp randomly in the interval [1, p.nl/2]
Initialize the set of transferable layers: Lnext := p.l
FOR i := 1 TO nlp
    Select layer lnext ∈ Lnext with maximum volume utilization
    Update descendants:
    o.nl := o.nl + 1; o.L := o.L ∪ {lnext}
    o.xcfree := o.xcfree - lnext.d; o.Bfree := o.Bfree\B(lnext)
    Update transferable layers; Lnext := Lnext\{lnext}
Use basic heuristic (mutation variant) on descendant o
OUTPUT (o)
```

Fig. 9. Algorithm of standard mutation.

maximum utilisation are transferred unchanged to the offspring. The number of these layers is determined randomly; at least one and at maximum 50% of the parent layers may be transferred.

2. *Layer extension*: After the layer transfer, the still incomplete mutant is extended by newly generated layers into a complete stowage plan through the basic heuristic in the variant *mutation*.

A random alteration of the parent solution is carried out solely in the layer extension phase by randomly selecting layer definition variants of new layers within the basic heuristic.

Filling the container layer-for-layer is in general combined with some losses of space capacity utilisation. The second *mutation* variant, the *merger mutation*, aims at reducing these losses. For this purpose, stowage plans are generated in which individual layers have a comparably larger depth.

In contrast to the *standard mutation*, a *merger mutation* includes the transfer of all parent layers – with the exception of two layers – to the mutant. One of these two layers is determined at random among all parent layers, while the other layer is selected at random from the 50% of those parent layers with the worst layer utilisations.

The layer extension proceeds analogously to the *standard mutation*. However, only a single additional layer is generated. The depth of this layer is determined as the total free container depth after layer transfer. The result of a *merger mutation* is therefore a stowage plan which contains one layer fewer than the parent stowage plan.

4.4. Configuration

The configuration of the GA covers the definition of its parameters and its termination criteria. For the parameters which have already been introduced, the following values are fixed:

- Population size $npop = 50$.
- Number of solutions to be reproduced per generation $nrep = 10$.
- Application probability for the *crossover* $p_{cross} = 0.67$.
- Application probability for the *standard mutation* $p_{mut} = 0.33$.
- Number of solutions to be generated through *merger mutation* per generation $nmerge = 10$.
- The percentages $qldb_1$, $qldb_2$ and $qldb_3$ indicate the proportion of layer definitions to be used when generating new layers with the help of the basic heuristic (cf. Section 3.1); depending on the number of box types *nbtype* in the problem to be solved, these parameter values are graduated as follows:
  - ○ $qldb_1 = 100$, $qldb_2 = 30$, $qldb_3 = 33$ if $nbtype \leqslant 20$,
  - ○ $qldb_1 = 30$, $qldb_2 = 10$, $qldb_3 = 10$ if $nbtype > 20$ and $nbtype \leqslant 50$,

  - ∘ $qldb_1 = 10$, $qldb_2 = 10$, $qldb_3 = 10$ if $nbtype > 50$ and $nbtype \leqslant 70$,
  - ∘ $qldb_1 = 5$, $qldb_2 = 5$, $qldb_3 = 5$ if $nbtype > 70$;
- rules (R3.1) and (R3.2) for determining the rotation variants when generating the placings for residual spaces (cf. Section 3.2) are applied alternatively; during the first 50% of the generations rule (R3.1) is used and rule (R3.2) is used during the last 50% of the generations.

Two termination criteria are used simultaneously for the GA. The search is terminated if:

- in addition to the *start* generation a number of $ngen = 500$ further generations has been generated or
- the time consumed exceeds the upper limit $maxtime = 500$ seconds; a corresponding time check is always performed at the end of the calculation of a generation.

The given parameter values have proved their suitability in multiple low-range tests. The continuous use of constant parameter values in the test calculations intends to demonstrate the robustness of the hybrid GA. Robustness in the sense of achieving an evenly high solution quality for a broad range of container loading problems in spite of uniform parameters. For the same reason, in all test calculations a uniform starting value, namely, 1, was used for the random number generation, and only one run was carried out per problem.

5. The inclusion of required constraints

The given constraints are to be considered under two aspects. Firstly, it must be guaranteed that generated solutions meet "hard" constraints. In addition, certain procedural modifications should ensure that even for constrained problems a high volume utilisation is achieved. The consideration of constraints (C1)–(C5) will now be described in detail.

(C1) *Orientation constraint*: In order to take account of an orientation constraint (C1) the selection of the rotation variant for a box placing is limited to those variants which are feasible with respect to the given problem.

(C2) *Stability constraint*: The stability constraint (C2), which concerns the full support of boxes not stowed directly on the floor of the container, is always respected without any further measures. The exclusion of placements with lateral overhanging boxes within the basic heuristic guarantees a full support (cf. Section 3.2).

(C3) *Stacking constraint*: In this case, a load transfer through additional placed boxes must be excluded. This is ensured as follows: On the one hand, a residual daughter-space of the type *above* is not generated for a box which has been placed in the reference corner of a residual space and which is subject to a stacking constraint. In addition, if two boxes are to be placed in a residual space, then it is excluded that a box which may bear no weight is placed in the reference corner of the residual space and the second box is placed on top of it.

In order to reduce the space loss caused by boxes which are subject to stacking constraints, two procedural modifications are introduced. These modifications aim at placing these boxes as high as possible in the container:

1. The first modification concerns the generation of layer definition lists. Layer definitions including a layer-defining box which may bear no weight are only accepted if the following condition holds: feasible layer definitions including a layer-defining box which is of larger volume and which may bear a weight cannot be generated.
2. The second modification concerns the placing rule (R1) for selecting box pairs to be filled in a residual space. The rate of boxes subject to stacking constraints is understood here as the volume share of these boxes of the total volume of the boxes to be stowed. If this rate is less or equal to 33%, then box pairs including a box subject to a stacking constraint are not accepted for residual spaces on the container floor. If the rate is above 33%, then box pairs of this type are only accepted for residual spaces on the container floor, if no placing at all can otherwise be found for the residual space. An analogue condition applies, if the rate of boxes subject to stacking constraints is above 33%

and residual spaces lie higher than one third of the container height $zc/3$. In this case, box pairs without a box subject to stacking constraints are only accepted if a placing cannot otherwise be found.

(C4) *Weight constraint*: In order to respect a weight constraint, a record is kept of the weight of the boxes already stowed in the container during the generation of a solution. Stowage of additional boxes is only permitted if the weight limit for the load is not exceeded.

(C5) *Balance constraint*: To take account of a balance constraint, the determined solution is reworked by means of an elementary heuristic. The heuristic manipulates the stowage plan by means of shifts of the load in x-direction combined with permutations of the layer sequence and reflections of some layers with regard to the plane $y = yc/2$ thus leading to a more balanced load. The heuristic is used in the GA from Gehring and Bortfeldt (1997) and is described there in greater detail.

While the remaining constraints can always be satisfied strictly, a balance constraint cannot always be satisfied in a strict sense. However, the mentioned heuristic usually improves the position of the centre of gravity of the load.

6. Test of the hybrid GA

The following test demonstrates the performance of the hybrid GA in comparison to other methods. In the following, the underlying test problems, the considered methods and the derived numerical results are described.

6.1. Test problems

The test is based on the following three groups of problems:

1. 15 problems introduced by Loh and Nee (1992);
2. 1500 problems suggested by Bischoff and Ratcliff (1995) and Davies and Bischoff (1998);
3. 150 problems from Bischoff and Ratcliff (1995) with additional constraints.

The problems from Loh and Nee may be classified as weakly heterogeneous throughout. On average, there are 22.2 boxes for each box type. The problems are denoted by LN01 to LN15. The only constraint that is explicitly required is an orientation constraint (C1).

The 1500 problems from Bischoff and Ratcliff are subdivided into 15 test cases each with 100 problem examples which are referred to as BR1 to BR15. The number of different box types corresponds in each test case. These numbers are given for the 15 test cases by 3, 5, 8, 10, 12, 15, 20, 30, 40, 50, 60, 70, 80, 90 and 100. According to the decreasing average number of boxes per box type, the problem character in the 15 test cases changes gradually from weakly heterogeneous to strongly heterogeneous. In test case BR1 there are on average 50.2 boxes for each box type, but in test case BR15 the average number is only 1.30. Again only an orientation constraint (C1) is to be met.

The 150 problems in group 3 are subdivided into 15 test cases each with 10 problems and they are denoted by BR1R to BR15R. Each test case BR*i*R, $i = 1, \ldots, 15$, includes exactly the first 10 problems of the respective test case BR*i* from Bischoff and Ratcliff. Each problem is extended by two additional constraints:

- The first additional constraint is a stacking constraint (C3). This requires that the boxes included in the first 50% of all box types of the original problems may bear no weight.
- The second additional constraint is a weight constraint (C4). On the one hand, constraint (C4) assumes that the weight of each box is determined proportionally to its volume. On the other hand, the total weight of the box set is set at 150% of the arbitrarily chosen maximum weight of the container load.

Insofar as methods from the authors are tested, each calculated problem is extended by a stability constraint (C2) and a "weak" balance constraint (C5). The balance constraint requires that the deviation of the load's centre of gravity from the middle of the container in both horizontal directions is not greater than 1% of the respective container side dimension.

6.2. Considered methods

Along with the presented hybrid GA – denoted by CBGAS – the following methods were included in the test:

- the heuristic from Loh and Nee (1992);
- the heuristic from Ngoi et al. (1994);
- the heuristic from Bischoff et al. (1995);
- the heuristic from Bischoff and Ratcliff (1995);
- the GA from Gehring and Bortfeldt (1997), denoted by CBGAT;
- the tabu search procedure from Bortfeldt and Gehring (1998), denoted by CBUSE.

The results of the first four methods are reported here as presented in the cited literature. It should be noted that the volume utilisations of the methods from Bischoff et al. (1995) and from Bischoff and Ratcliff (1995) for the 1500 problems from Bischoff and Ratcliff are listed in accordance with the paper by Davies and Bischoff (1998).

The results of the methods CBGAS, CBGAT and CBUSE – throughout implemented in C – were calculated on a Pentium PC with a frequency of 400 MHz. The GA CBGAT and the CBUSE tabu search procedure were parameterised as shown in Gehring and Bortfeldt (1997) or Bortfeldt and Gehring (1998). The time limit *maxtime* = 500 seconds was also used for CBGAT and CBUSE.

6.3. Test results

1. *Results for the* 15 *problems from Loh and Nee* (1992): Table 1 shows the test results obtained for the 15 problems from Loh and Nee. For each method and problem the number of unstowed boxes is shown along with the achieved volume utilisation where this number is greater than zero. Finally, the average volume utilisation over all 15 problems is given per method. Note that Loh and Nee (1992) use a capacity utilisation criterion referred to as "packing density", which overestimates the usual volume utilisation (cf. Bischoff and Ratcliff, 1995).

To sum up, it can be stated that the hybrid GA CBGAS does not achieve the solution quality of the tabu search procedure CBUSE when applied

Table 1
Results obtained for the 15 problems from Loh and Nee (1992)

Problems	Packing density (%); Loh and Nee (1992)	Volume utilisation (%)					
		Ngoi et al. (1994)	Bischoff et al. (1995)	Bischoff and Ratcliff (1995)	Gehring and Bortfeldt (1997) (CBGAT)	Bortfeldt and Gehring (1998) (CBUSE)	Bortfeldt and Gehring (CBGAS)
LN01	78.1	62.5	62.5	62.5	62.5	62.5	62.5
LN02	76.8 (32)	80.7 (54)	89.7 (23)	90.0 (35)	90.7 (36)	96.7 (28)	89.8 (51)
LN03	69.5	53.4	53.4	53.4	53.4	53.4	53.4
LN04	59.2	55.0	55.0	55.0	55.0	55.0	55.0
LN05	85.8	77.2	77.2	77.2	77.2	77.2	77.2
LN06	88.6 (45)	88.7 (48)	89.5 (24)	83.1 (77)	91.1 (36)	96.3 (40)	92.4 (45)
LN07	78.2 (21)	81.8 (10)	83.9 (1)	78.7 (18)	82.7 (5)	84.7	84.7
LN08	67.6	59.4	59.4	59.4	59.4	59.4	59.4
LN09	84.2	61.9	61.9	61.9	61.9	61.9	61.9
LN10	70.1	67.3	67.3	67.3	67.3	67.3	67.3
LN11	63.8	62.2	62.2	62.2	62.2	62.2	62.2
LN12	79.3	78.5	76.5 (3)	78.5	78.5	78.5	78.5
LN13	77.0 (15)	84.1 (2)	82.3 (5)	78.1 (20)	85.6	85.6	85.6
LN14	69.1	62.8	62.8	62.8	62.8	62.8	62.8
LN15	65.6	59.6	59.5	59.5	59.5	59.5	59.5
Average value	**74.2**	**69.0**	**69.5**	**68.6**	**70.0**	**70.9**	**70.1**

to the problems from Loh and Nee. However, with regard to the average volume utilisation and the number of optimum solutions achieved, CBGAS performs better than the other methods.

2. *Results for the* 1500 *problems from Bischoff and Ratcliff* (1995): The results for test cases BR1 to BR15 are shown in Table 2. The average achieved volume utilisation and – in brackets – the dispersion (S.D.) as a percentage of the container volume are shown for each test case and method. Finally, the average volume utilisation over all 1500 problems is given for all methods.

The hybrid GA CBGAS achieves a clearly higher volume utilisation for all 15 test cases BR1 to BR15 than the GA CBGAT and the methods from Bischoff et al. (1995) and Bischoff and Ratcliff (1995).

The comparison of CBGAS with the tabu search algorithm CBUSE indicates that CBGAS is tailored to strongly heterogeneous problems. While CBUSE achieves better results in the first five test cases BR1 to BR5, CBGAS achieves a higher average volume utilisation in the 10 test cases BR6 to BR15. In the case of CBUSE, the average volume utilisation per test case falls monotonically with the increase in the problem heterogeneity. In contrast, the average volume utilisations calculated with CBGAS increase up to test case BR5. Averaged over all 15 test cases CBGAS achieves a higher volume utilisation of about 1.5%.

The average computing time per problem amounts to 316.0 seconds for CBGAS and 242.6 seconds for CBUSE, whereas the second GA, CBGAT, requires only 11.7 seconds per problem. For the remaining methods computing times have not been reported.

With regard to the stability of the load it can be stated for all compared methods that in each problem every stowed box is supported 100% from below. The stability constraint (C2) is therefore generally met. However, the results differ with regard to two further stability criteria, namely, the average number of supporting boxes and the gap stability. The corresponding results for two of the tested methods are taken from the paper by Bischoff and Ratcliff (1995); the values for these stability criteria refer therefore generally to the first 700 of the 1500 problems from Bischoff and Ratcliff.

The average number of boxes which support a box not placed on the container bottom should be

Table 2
Results for test cases BR1 to BR15 from Bischoff and Ratcliff (1995)

Test cases (No. box types per problem)	Volume utilisation (%)		Volume utilisation/dispersion (%)		
	Bischoff et al. (1995)	Bischoff and Ratcliff (1995)	Gehring and Bortfeldt (1997) (CBGAT)	Bortfeldt and Gehring (1998) (CBUSE)	Bortfeldt and Gehring (CBGAS)
BR1 (3)	81.76	83.37	86.77 (4.07)	92.63 (2.61)	87.81 (3.30)
BR2 (5)	81.70	83.57	88.12 (2.76)	92.70 (1.81)	89.40 (2.34)
BR3 (8)	82.98	83.59	88.87 (2.23)	92.31 (1.51)	90.48 (1.52)
BR4 (10)	82.60	84.16	88.68 (1.97)	91.62 (1.49)	90.63 (1.42)
BR5 (12)	82.76	83.89	88.78 (1.79)	90.86 (1.20)	90.73 (1.27)
BR6 (15)	81.50	82.92	88.53 (1.58)	90.04 (1.18)	90.72 (1.18)
BR7 (20)	80.51	82.14	88.36 (1.28)	88.63 (1.27)	90.65 (1.05)
BR8 (30)	79.65	80.10	87.52 (1.18)	87.11 (1.40)	89.73 (1.07)
BR9 (40)	80.19	78.03	86.46 (1.42)	85.76 (1.35)	89.06 (0.98)
BR10 (50)	79.74	76.53	85.53 (1.18)	84.73 (1.35)	88.40 (0.97)
BR11 (60)	79.23	75.08	84.82 (1.34)	83.55 (1.49)	87.53 (1.16)
BR12 (70)	79.16	74.37	84.25 (1.36)	82.79 (1.44)	86.94 (1.08)
BR13 (80)	78.23	73.56	83.67 (1.48)	82.29 (1.49)	86.25 (1.11)
BR14 (90)	77.40	73.37	82.99 (1.55)	81.33 (1.47)	85.55 (1.16)
BR15 (100)	75.15	73.38	82.47 (1.48)	80.85 (1.30)	85.23 (1.05)
Average value	**80.17**	**79.20**	**86.39**	**87.15**	**88.61**

as high as possible for reasons of stability. Averaged over the first 700 problems, this number is 1.05 for the GA CBGAS. More favourable average values, namely, 2.08 and 1.29, respectively, are achieved with the heuristic from Bischoff et al. and with the tabu search algorithm CBUSE.

The criterion of gap stability measures the occurrence of cavities in the load. For a given stowage plane the gap stability is defined as the percentage of boxes with insufficient side support in relation to all stowed boxes. A box is deemed to have insufficient side support, if it does not adjoin another object (box, container wall) on at least three sides. The GA CBGAS achieves an average gap stability rate of 30.85% for the first 700 problems. The heuristic from Bischoff et al. and the GA CBGAT achieve considerably better values with 17.71% and 22.46%, respectively.

3. *Results for the* 150 *problems with additional constraints*: The test results for the 150 extended problems from Bischoff and Ratcliff are only summarised in the following. Results are available only for the procedures CBGAS, CBGAT and CBUSE.

Measured over all 15 test cases, the methods CBGAS, CBGAT and CBUSE achieve volume utilisation rates of 65.44%, 64.39% and 63.79%, respectively. CBUSE dominates for the first 4 test cases with weakly heterogeneous problems only, and CBGAS achieves the best performance for the other 11 test cases. It should be noted that – due to the weight constraint (C4) – only two-thirds of the total box volume can be stowed at maximum. For this reason, all three procedures, in particular CBGAS, achieve good volume utilisation rates for the more strongly constrained 150 problems.

Finally, results considering the weak balance constraint (C5) are presented. Averaged over the test cases BR1 to BR15 the deviations of the load centre from the middle of the container obtained for the GA CBGAT amount to 1.36% of the respective container side dimension for both horizontal directions. For the tabu search algorithm CBUSE the averaged percentage deviation of the load centre from the container middle is 0.97% of the respective container side dimension. Finally, the corresponding value for the GA CBGAS measured over test cases BR1 to BR15 is only 0.10%. For a given container depth of 587 cm, this means an average absolute deviation of less than 1 cm in the length of the container. For the more strongly constrained test cases BR1R to BR15R the GA CBGAS achieved an average deviation which was almost as low at 0.26%.

7. Concluding remarks

The presented hybrid GA for loading a single container is particularly suitable for strongly heterogeneous problems. The algorithm takes account of some constraints which are relevant in practice. Its good performance and superiority to comparative methods was verified in an extensive test. The use of a time limit kept the required computing times within acceptable limits.

The algorithm includes a hybridisation concept which was recommended by Michalewicz (1992). According to this concept, solutions are represented by means of more complex data structures and manipulated by means of problem-specific operators. The dominance of the algorithm over the GA from Gehring and Bortfeldt (1997), which is based on a more traditional development concept for GAs, indicates the suitability of the problem-specific approach for the field of container loading problems.

The presented algorithm and the tabu search algorithm from Bortfeldt and Gehring (1998) behave roughly in a complementary manner with regard to volume utilisation. The tabu search achieves higher volume utilisation rates for weakly heterogeneous box sets, while the presented algorithm dominates for strongly heterogeneous box sets. Comparisons with other methods display a similarly complementary behaviour with regard to given constraints and desired stowage plan characteristics. The advantages and disadvantages of the different tested methods are caused primarily by the heuristic stowing concept, and not by the applied search strategy. This appears to support the opinion that various heuristic approaches for stowing are required, one of which has to be selected depending on the special conditions and requirements of the underlying practical application.

References

Bischoff, E.E., Janetz, F., Ratcliff, M.S.W., 1995. Loading pallets with non-identical items. European Journal of Operational Research 84, 681–692.

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

Bortfeldt, A., 1994. A genetic algorithm for the container loading problem. In: Proceedings of the Conference on Adaptive Computing and Information Processing, London, vol. 2, pp. 749–757.

Bortfeldt, A., Gehring, H., 1998. Ein Tabu Search-Verfahren für Containerbeladeprobleme mit schwach heterogenem Kistenvorrat. Operations Research Spektrum 20, 237–250.

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

Davis, L. (Ed.), 1991. Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York.

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

Falkenauer, E., 1998. Genetic Algorithms and Grouping Problems. Wiley, Chichester, UK.

Gehring, H., Menschner, K., Meyer, M., 1990. A computer-based heuristic for packing pooled shipment containers. European Journal of Operational Research 44, 277–288.

Gehring, H., Bortfeldt, A., 1997. A genetic algorithm for solving the container loading problem. International Transactions of Operational Research 4 (5/6), 401–418.

Gehring, H., Schütz, G., 1994. Zwei genetische Algorithmen zur Lösung des Bandabgleichproblems. In: Werners, B., Gabriel, R. (Eds.), Operations Research, Reflexionen aus Theorie und Praxis. Springer, Berlin, pp. 85–128.

Kopfer, H., Pankratz, G., Erkens, E., 1994. Entwicklung eines hybriden genetischen Algorithmus zur Tourenplanung. Operations Research Spektrum 16, 21–31.

Lin, J.-L., Foote, B., Pulat, S. Chang, C.-H., Cheung, J.-Y., 1993. Hybrid genetic algorithm for container packing in three-dimensions. In: Proceedings of the Ninth IEEE Conference on Artificial Intelligence. IEEE Computer Soc. Press, Washington, DC, pp. 353–358.

Loh, T.H., Nee, A.Y.C., 1992. A packing algorithm for hexahedral boxes. In: Proceedings of the Conference of Industrial Automation, Singapore, pp. 115–126.

Michalewicz, Z., 1992. Genetic Algorithms + Data Structures = Evolution Programs. Springer, Berlin.

Morabito, R., Arenales, M., 1994. An AND/OR-graph approach to the container loading problem. International Transactions of Operational Research 1 (1), 59–73.

Ngoi, B.K.A., Tay, M.L., Chua, E.S., 1994. Applying spatial representation techniques to the container packing problem. International Journal of Production Research 32, 111–123.

Portmann, M.C., 1990. An efficient algorithm for container loading. Methods of Operations Research 64, 563–572.

Prosser, P., 1988. A hybrid genetic algorithm for container loading. In: Pradig, B. (Ed.), Proceedings of the Eighth European Conference on Artificial Intelligence. Pitman, London, pp. 159–164.

Reeves, C.R. (Ed.), 1993a. Modern Heuristic Techniques for Combinatorial Problems. Blackwell, Oxford.

Reeves, C.R., 1993b. Genetic algorithms and combinatorial optimization. In: Proceedings of the Conference on Adaptive Computing and Information Processing. Part 2, London, pp. 711–723.

Scheithauer, G., 1992. Algorithms for the container loading problem. In: Operational Research Proceedings. Springer, Berlin, 1991, pp. 445–452.

Sixt, M., 1996. Dreidimensionale Packprobleme. Lösungsverfahren basierend auf den Metaheuristiken Simulated Annealing und Tabu-Suche, Peter Lang – Europäischer Verlag der Wissenschaften, Frankfurt am Main.

Whitley, D., 1989. The GENITOR algorithm and selective pressure: why rank-based allocation of reproductive trials is best. In: Schaffer, J.D. (Ed.), Proceedings of the Third International Conference on Genetic Algorithms. Morgan Kaufmann, Los Altos, CA, pp. 133–140.