Available online at www.sciencedirect.com

SCIENCE DIRECT®

European Journal of Operational Research 172 (2006) 814–837

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/ejor

Discrete Optimization

A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces

Andreas Bortfeldt *

Department of Information Systems, University of Hagen, Hagen, Germany

Received 31 October 2003; accepted 25 November 2004
Available online 22 January 2005

Abstract

Given a set of rectangular pieces and a container of fixed width and variable length, the two-dimensional strip packing problem (2D-SPP) consists of orthogonally placing all the pieces within the container, without overlapping, such that the overall length of the layout is minimised. Until now mainly heuristics, for example genetic algorithms (GA), were proposed for the 2D-SPP which use encoded solutions that are manipulated by standard operators. In this paper a GA for the 2D-SPP is suggested that works without any encoding of solutions. Rather fully defined layouts are manipulated as such by means of specific genetic operators. Two additional constraints, namely the orientation constraint and the guillotine constraint, can be taken into account. The GA is subjected to a comprehensive test using benchmark instances with up to 5000 pieces. Compared to eleven competing methods from the literature the GA performs best.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Packing; Two-dimensional strip packing problem; Rectangular pieces; Genetic algorithm

1. Introduction

The subject of this paper is the two-dimensional strip packing problem with a set of rectangularly shaped pieces and a larger rectangle with a fixed width and variable length, designated as the container. The search is for a feasible layout of all the pieces in the container that minimises the required container length and, where necessary, takes additional constraints into account.

A layout is considered feasible if pieces do not overlap, all pieces lie within the container and are arranged orthogonally, i.e. parallel to the container edges. Additionally, two constraints are optionally included in the problem:

* Address: FernUniversität Hagen, Lehrstuhl Wirtschaftsinformatik, Profilstrasse 8, 58084 Hagen, Germany. Tel.: +49 2331 9874431.
E-mail address: andreas.bortfeldt@fernuni-hagen.de (A. Bortfeldt).

(C1) Orientation constraint

While rotating the pieces by 90° is permitted in general, the orientation of all pieces is fixed by the orientation constraint.

(C2) Guillotine constraint

This constraint requires that all pieces placed in a layout can be reproduced by a series of guillotine cuts, i.e. edge-to-edge cuts parallel to the edges of the container.

The strip packing problem (SPP) can be considered both in two (2D-SPP) and in three dimensions. Together with the container loading problem and the bin packing problem, the SPP represents a further basic type of more-dimensional packing and cutting problems (cf. Coffman and Shor, 1990; Martello et al., 1998) with considerable practical relevance. The 2D-SPP occurs, e.g., in the cutting of rolls of paper or metal. In the three-dimensional case the solution of strip packing problems can improve the design of containers or even help the selection of vehicles in a vehicle fleet for transporting a given volume of goods (cf. Sixt, 1996).

Taking into account the constraints (C1) and (C2) the following four subtypes of the strip packing problem can be distinguished (cf. Lodi et al., 1999):

- RF: the pieces may be rotated by 90° (R) and no guillotine cutting is required (F);
- RG: the pieces may be rotated by 90° (R) and guillotine cutting is required (G);
- OF: the orientation of the pieces is fixed (O) and no guillotine cutting is required (F);
- OG: the orientation of the pieces is fixed (O) and guillotine cutting is required (G).

Obviously, a feasible solution with regard to the subtype OG is also feasible in terms of the three other subtypes of the SPP and a feasible solution with regard to the subtypes RG and OF, respectively, is also feasible in terms of the subtype RF.

The SPP is NP-hard (cf. Hopper and Turton, 2001). Only few exact approaches for the 2D-SPP are known so far and their applicability is limited to piece sets with up to 200 pieces. Martello et al. (2003), e.g., propose a branch-and-bound method for the 2D-SPP while Fekete and Schepers (1997) develop a general framework for the exact solution of more-dimensional packing problems.

In the literature heuristics have been deemed most suitable for solving the 2D-SPP. These are, to an increasing degree metaheuristics, mainly genetic algorithms (GA), but simulated annealing (SA), tabu search (TS) and other types of metaheuristics are also applied. Hopper and Turton (2000, 2001) provide an extensive up-to-date overview of the metaheuristics that have been developed for the different variants of the 2D-SPP. To this end, Hopper and Turton differentiate between the following three groups of metaheuristic solution approaches:

(1) The methods of the first group use a coding of solutions. Typically, an encoded solution stipulates a placement sequence for the pieces. The search is carried out by the respective metaheuristic in the space of the encoded solutions and usually uses problem-independent operators. A placement or decoding routine serves to transform encoded solutions into complete layouts. An example of a placement routine is given by the so-called bottom-left (BL) heuristic. If a horizontally placed container is assumed, the bottom-left routine displaces each piece to be placed in the container alternately in two horizontal, orthogonal directions, until displacement is no longer possible (cf. Hopper and Turton, 2000).

(2) Solution approaches for the second group have an intermediate position. While on the one hand encoded solutions already contain, to a certain extent, layout or geometrical information, an additional placement routine is also required for the final positioning. Typical for this group is a problem-specific coding, which is often based on graphs, and corresponding problem-specific operators.

(3) The approaches in group 3 do not use coding. The search takes place directly in the space of the fully defined layouts, which are therefore manipulated as such by specific operators.

In Table 1 a representative sample of recently developed metaheuristics for the 2D-SPP with

Table 1
Metaheuristics for the 2D-SPP with rectangular pieces

No.	Authors, source	SPP subtype	Group of approaches	Type of metaheuristic	Additional remarks
1	Jakobs (1996)	RF	1	GA	• Uses the BL heuristic as decoder
2	Dagli and Poshyanonda (1997) and Poshyanonda and Dagli (2004)	RF	1	GA	• Placement routine is based on a sliding method and makes use of an artificial neural network
3	Liu and Teng (1999)	RF	1	GA	• Uses an improved BL heuristic that gives priority to width-orthogonal shifting of a piece
4	Hopper and Turton (2000)	RF	1	GA	• No. 4–6 are the most successful methods proposed in Hopper and Turton (2000)
5	Hopper and Turton (2000)	RF	1	SA	• Each of these methods uses the bottom left fill (BLF) heuristic that is capable of filling gaps in a layout
6	Hopper and Turton (2000)	RF	1	NE	• NE stands for naive evolution, i.e. a GA with mutation but without crossover
7	Mumford-Valenzuela et al. (2003)	RG	1	GA	• Based on a normalised postfix representation that offers a unique encoding for each feasible layout • Uses standard operators such as cycle crossover
8	Kröger (1993, 1995)	RG	2	Parallel GA	• A solution is encoded as a directed binary tree that fixes one dimension of the position of each piece • The second dimension is fixed by the placement routine
9	Schnecke (1996)	RG	2	Parallel GA	• Similar to no. 8
10	Ratanapan and Dagli (1997, 1998)	RF	3	GA	• Specific genetic operators conduct geometrical operations such as translation and rotation of pieces
11	Iori et al. (2002) and Monaci (2001)	OF	Hybrid	Hybrid, combines GA and TS	• GA: solutions are encoded as placement sequences • TS: derived from the TS method by Lodi et al. (1999) for the 2D bin packing problem • Additional procedures for generating initial solutions or for post-optimising solutions look at layouts directly

rectangular pieces is overviewed. Each of the metaheuristics is characterised by the covered SPP subtype, the respective group of solution approaches (as introduced above), the type of the metaheuristic and some additional remarks. For further information the reader is referred to the literature.

A genetic algorithm for the 2D-SPP that covers each of the SPP subtypes will be presented below. It is obtained by adapting the GA from Bortfeldt and Gehring (2001), which was developed for the 3D container loading problem (CLP) with a single container to be loaded.

The GA for the 2D-SPP represents a solution approach of group 3, which has been seldom used so far. A simpler version of the GA was presented in Bortfeldt and Gehring (1999). From a methodological point of view, it is the intention of this paper to demonstrate that a genetic search without any encoding of solutions is suitable to tackle the 2D strip packing problem.

The remainder of the paper is organised as follows: Section 2 provides a description of the original GA for the container loading problem. Section 3 shows the adaptation of the GA to the 2D-SPP; Section 4 subjects the GA for the 2D-SPP to a benchmark test and Section 5 contains a summing-up of the paper.

2. The genetic algorithm for the container loading problem

In the following the GA from Bortfeldt and Gehring (2001) for solving the container loading problem (in short CLP-GA) is described. The reader is referred, e.g., to Falkenauer (1998) for an introduction into genetic algorithms. In accordance with the subject of the paper at hand a 2D container loading problem is assumed. It can be formulated as follows: for a set of rectangular pieces and a rectangular container with fixed dimensions determine a layout including a subset of the pieces such that the selected pieces are orthogonally placed within the container, without overlapping, and the covered area of the container is maximised.

2.1. Geometrical structure of layouts

The GA exclusively generates layouts with a layer structure. Each layout consists of successive rectangular layers in which one or more pieces are arranged. Each piece belongs to exactly one layer. The width of a layer corresponds to the container width. The layer depth is stipulated by a so-called layer-determining piece (*ldp*) and its orientation. The structure of layouts is illustrated in

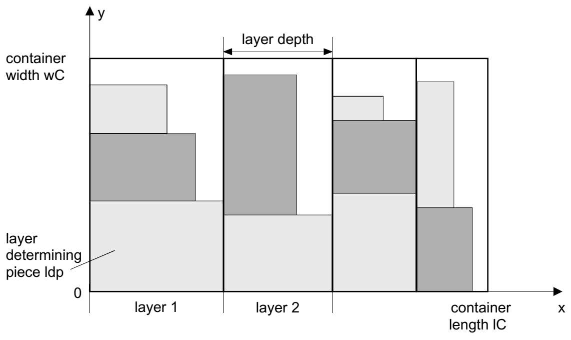


Fig. 1. Layer structure of a layout.

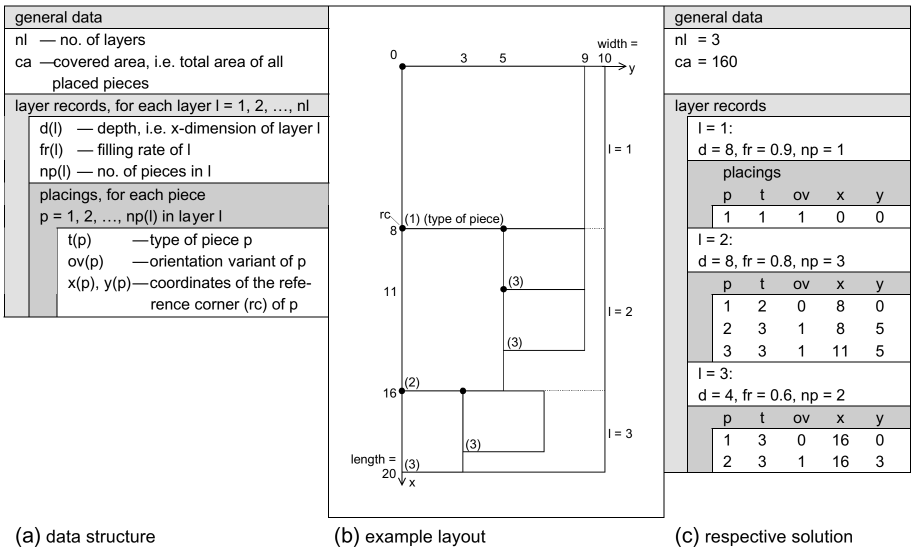
Fig. 1; this also shows the embedding of the container in the 2D coordinate system used here.

2.2. Representing layouts and determining their fitness

The genetic search is carried out directly in the space of the completely defined layouts with a layer structure. The representation of layouts, i.e. the data structure for solutions of the GA, is defined and illustrated by an example in Fig. 2.

Some elements of the representation need a further clarification:

- The type of a piece is given by its dimensions, i.e. the pieces of a type represent congruent rectangles. The piece types are numbered in descending order according to the piece area. Subsequently, it sometimes will not explicitly be differentiated between pieces and their types.
- The orientation of a piece *p* in a layout is given by an orientation variant *ov*(*p*); *ov*(*p*) is equal to 0 if the shorter edge of *p* lies in parallel to the container width and equal to 1 otherwise.
- The reference corner of a piece *p* in a layout designates the corner of *p* that is nearest to the origin of the 2D coordinate system.
- The type *t*(*p*) of a piece *p*, its orientation variant *ov*(*p*) and the respective coordinates *x*(*p*), *y*(*p*) of the reference corner represent a so-called placing within a layer.
- The filling rate of a layer is calculated as the quotient of the total area of all pieces



general data
nl — no. of layers
ca —covered area, i.e. total area of all placed pieces
layer records, for each layer l = 1, 2, …, nl
d(l) — depth, i.e. x-dimension of layer l
fr(l) — filling rate of l
np(l) — no. of pieces in l
placings, for each piece p = 1, 2, …, np(l) in layer l
t(p) —type of piece p
ov(p) —orientation variant of p
x(p), y(p)—coordinates of the reference corner (rc) of p

general data
nl = 3
ca = 160

layer records

l = 1:
d = 8, fr = 0.9, np = 1

placings

p	t	ov	x	y
1	1	1	0	0

l = 2:
d = 8, fr = 0.8, np = 3

p	t	ov	x	y
1	2	0	8	0
2	3	1	8	5
3	3	1	11	5

l = 3:
d = 4, fr = 0.6, np = 2

p	t	ov	x	y
1	3	0	16	0
2	3	1	16	3

(a) data structure (b) example layout (c) respective solution

Fig. 2. Representation of layouts.

placed in the layer and the layer area. The substructure of all layer data is called a layer record.

- The layers of a solution are always arranged in descending order according to the filling rate, i.e. the layer next to the origin of the coordinate system has the highest filling rate. After a solution has been constructed its layers are rearranged and the x-coordinates of the placings are adapted if necessary.
- The objective function value of a solution, i.e. the covered part of the container area, serves as the fitness of the solution.

Obviously, a solution completely determines the corresponding layout. Thus, a decoding of solutions is not necessary.

2.3. Setting-up the genetic search

The overall procedure of the CLP-GA is shown in Fig. 3.

Some essential characteristics of the GA should be stressed:

- The GA uses the reproduction model of generation-wise replacement and the size of the population, given by a parameter *npop*, is held constant during the search. No duplicates are permitted within the start generation or in the subsequent generations. The search is aborted immediately after a solution was found that includes all pieces (indicating an optimal solution).
- In order to create a new generation initially the *nrep* (*nrep* $\geqslant 1$) best solutions of the former generation are reproduced (elitist strategy). After this, the new generation is filled up to the full size *npop* by generating solutions through crossover and the first mutation variant, called standard mutation. Both operators are used alternatively with the constant and complementary probabilities *pcross* and *pmut* ($pcross + pmut = 1$). Finally, *nmerge* additional solutions are generated through the second mutation variant, called merger mutation. These solutions replace, where necessary, poorer solutions from the new generation.
- Parent solutions for the crossover and both mutation variants are chosen by means of a ranking selection. However, the second crossover partner is selected purely at random.

The problem-specific operators, namely the crossover and both mutation variants, proceed in two steps:

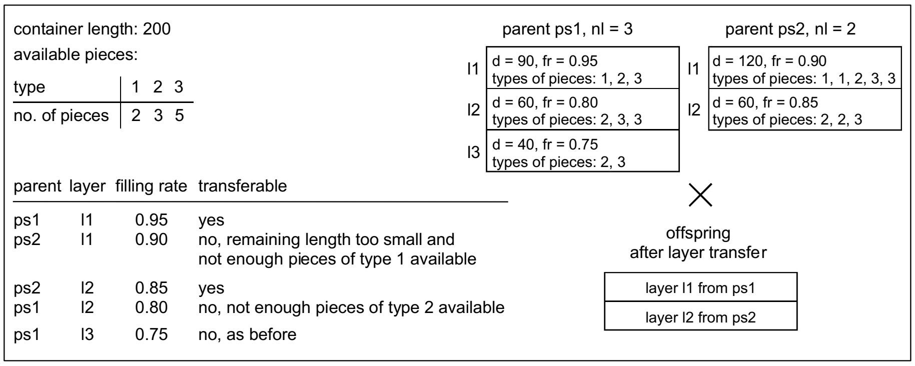
```
procedure cl_genetic_search(in: container length lC, container width wC, piece set P, out: best solution sbest)
set generation counter g := 0;
generate npop solutions for the start generation g;
for g := 1 to ngen do
        reproduce the best nrep solutions from generation g – 1 for generation g;
        while generation g contains fewer than npop solutions do
                if randomly chosen operator is crossover then
                        select parent solutions ps1, ps2 in generation g – 1;
                        generate offspring os by crossover for generation g;
                else {randomly chosen operator is mutation}
                        select parent solution ps in generation g – 1;
                        generate offspring os by standard mutation for generation g;
                endif;
        endwhile;
        for i := 1 to nmerge do
                select parent solution ps in generation g – 1;
                generate offspring os by merger mutation;
                if os is better than the worst solution ws in generation g then
                        replace ws by os;
                endif;
        endfor;
endfor;
end.
```

Fig. 3. Overall procedure of the GA for the CLP.

- At first the offspring is initialised to empty (0 layers) and some of the best layers in terms of the filling rates of the parent solution(s) are transferred unchanged to the offspring.
- After the layer transfer, the offspring can usually still be extended by further layers. Hence, in the second step the offspring is completed by newly generated layers.

The details of the layer transfer depend on the operator:

- If a crossover is carried out all the layers of both parents are examined in descending order according to the filling rate. The best parent layer is always transferred. A second, third etc. parent layer is only transferred if its depth does not exceed the residual container length and if the total number of existing pieces per type is respected. Fig. 4 gives an example for the layer transfer in the course of a crossover operation. Only those data of the parents are indicated which control the transfer. Nevertheless, complete layer records are transferred to the offspring.
- In the course of a standard mutation a number *nlt* of layers to be transferred is selected purely at random; at maximum 50% of the parent



container length: 200

available pieces:

type	1	2	3
no. of pieces	2	3	5

parent	layer	filling rate	transferable
ps1	l1	0.95	yes
ps2	l1	0.90	no, remaining length too small and not enough pieces of type 1 available
ps2	l2	0.85	yes
ps1	l2	0.80	no, not enough pieces of type 2 available
ps1	l3	0.75	no, as before

Fig. 4. Layer transfer within a crossover.

layers may be transferred. The best *nlt* layers of the parent are then transferred to the offspring.
- A merger mutation transfers all but two randomly selected layers of the parent to the offspring.

An offspring is completed by means of the procedure *complete_solution* that is described afterwards. This procedure also serves to create the start generation.

2.4. Completing a solution

A single new layer is generated in two steps:

- Initially a feasible layer variant consisting of a layer-determining piece *ldp* and an orientation variant *ov* is chosen. A layer variant is only feasible if the piece *ldp* is still free (not yet placed) and it can still be placed within the container in the orientation *ov*. In general the depth *d* of the layer is given by the dimension of *ldp* which runs parallel to the container length. But for the lastly generated layer of a solution the depth *d* is determined by the residual container length.
- In the second step the layer is filled by means of the procedure *fill_layer* explained below.

The procedure *complete_solution* generates a set of complete, i.e. no longer extendable, solutions *S_out* from one incomplete solution *s_in*. It can be outlined as follows:

- At first a set *Lv1* of feasible layer variants for the first new layer of *s_in* is determined. For each variant in *Lv1* a layer is generated and *s_in* is extended *alternatively* by each of these layers. This results in a set *S1* of temporary solutions.
- Each of the temporary solutions $s \in S1$ is then extended *separately* by additional layers. If no further layer can be added, *s* is inserted in the set *S_out* that is originally empty.
- Each time an additional layer for a solution *s* is to be created, a set *Lvn* of feasible layer variants with regard to the present state of *s* is determined. For each variant in *Lvn* a layer is generated experimentally and the layer with the highest filling rate is then added to *s*.
- Let be *ns* the desired number of complete solutions (cf. Table 2). At last the set *S_out* is reduced to the *ns* best solutions in terms of the covered area.

On the one hand the procedure is used to complete an offspring of a crossover and a mutation, respectively; on the other hand it serves to create the start generation. Hence, three use cases exist which are defined in Table 2.

The restrictions on the permitted layer variants are based on a descending sorting of all feasible layer variants in accordance with the *ldp* area. For example: the first new layer for a crossover offspring is to be generated, 20 feasible layer variants exist and $qldp_1$ is set to 50%. Then only the first 10 layer variants with largest *ldp* area are permitted, i.e. considered for *Lv1*.

The parameters $qldp_1$ and $qldp_2$ serve the control of the trade-off between the solution quality and the effort for a crossover operation while $qldp_3$

Table 2
Use cases of the procedure complete_solution

Characteristic	Use case: Generation of the start population	Generation of an offspring through crossover	Generation of an offspring through mutation (both variants)
No. of layers in *s_in*	0 layers	⩾1 layer	⩾1 layer
No. of desired complete solutions *ns*	Population size *npop*	1, i.e. only best solution	1, i.e. only best solution
Restriction on permitted variants with first layer (*Lv1*)	None	Only the first $qldp_1$% of all variants is permitted	Only one variant is permitted which is selected randomly among the first $qldp_3$% of all variants
Restriction on permitted variants with subsequent layers (*Lvn*)	Only first variant permitted	Only the first $qldp_2$% of all variants is permitted	As with first new layer

merely defines a selection range for layer variants that is used for mutations.

In a merger mutation only one layer is newly generated that replaces two selected parent layers (see above). In this way a merger mutation aims at reducing losses at the border between two layers.

2.5. Filling a layer

In order to fill a layer the responsible procedure *fill_layer* generates a set of placings. A residual space represents a free rectangular space within the layer. It is characterised by its side dimensions and its reference corner, i.e. the corner closest to the origin of the coordinate system. Each placing is established by arranging a piece with fixed orientation in the reference corner of a residual space. After a piece has been placed in a residual space, two daughter residual spaces (in short daughter spaces) are generated within it, which lie beside and in front of the piece. The two possible variants of the generation of daughter residual spaces are illustrated in Fig. 5.

The procedure *fill_layer* is shown in Fig. 6. In the following some details are highlighted:

- The procedure takes as input the layer-determining data (ldp, ov, d) and the set *Pfree* of still available pieces that is updated each time a piece has been placed. The placings are collected in a layer record L which is returned at last.

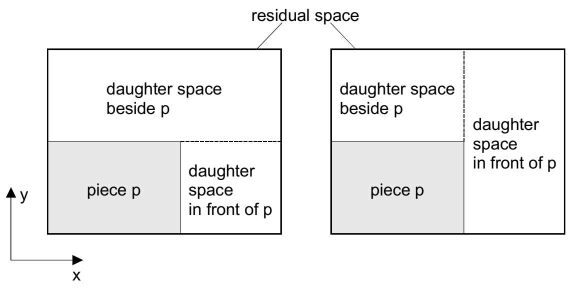


Fig. 5. Variants of the generation of daughter residual spaces.

```
procedure fill_layer(in: layer defining piece ldp, orientation ov, depth d, set of free pieces Pfree, out: layer record L)
initialise layer record L to empty;
initialise space stack SStack:
        - insert layer rectangle $d \times wC$ as single residual space;
        - record placing ldp, ov for this residual space;
$Pfree := Pfree \setminus \{ldp\}$;
while $SStack \neq \emptyset$ do
        set current residual space scurr to the uppermost element in SStack and remove this element from SStack;
        if scurr does include a placing then
                generate two daughter spaces within scurr in accordance with the present placing;
                insert the daughter spaces into SStack;
                add placing in scurr to layer record L;
        elseif at least one free piece can be placed in scurr then
                determine the pair of free pieces with maximal total area that fits completely in scurr;
                if only one piece was found then
                        determine orientation ov(pc) of the found piece pc;
                        generate two daughter spaces within scurr;
                        $Pfree := Pfree \setminus \{pc\}$;
                else {two pieces were found}
                        determine:
                        - piece pc for reference corner rc of scurr, orientation ov(pc),
                        - relative position of the other piece ps (infront of or besides pc), orientation ov(ps);
                        generate two daughter spaces within scurr taking into account the relative position of ps;
                        record placing ps, ov(ps) for the daughter space corresponding to the relative position of ps;
                        $Pfree := Pfree \setminus \{pc, ps\}$;
                endif;
                insert the daughter spaces into SStack;
                add placing pc, ov(pc), rc in scurr to layer record L;
        endif;
endwhile;
end.
```

Fig. 6. Procedure *fill_layer*.

- The layer rectangle acts as the first residual space. It is filled by the layer defining piece in the prescribed orientation (first placing) and then inserted into the stack *SStack* for residual spaces.
- Each time the following loop is passed through, the uppermost residual space of *SStack* is removed and processed (as *scurr*). Filling a layer terminates when the stack *SStack* is empty.
- If the current residual space *scurr* already includes a placing only the daughter spaces are generated and inserted as empty spaces in *SStack*. Moreover, the placing is accepted for the layer record. This situation occurs for the first residual space as well as for daughter spaces which are filled just after their generation (see below).
- If *scurr* is still empty and not loadable it is disregarded. Otherwise the pair of pieces with the maximum total area that can be placed completely in *scurr* is determined. A single piece (degenerated pair) is also admissible.
- If only one piece was found it is placed in *scurr* and the placing is taken over in the layer record. The daughter spaces are generated and inserted as empty spaces into *SStack*.
- If two pieces were found one piece (*pc*) is placed in the reference corner of *scurr* and this placing is again accepted for the layer record. The other piece (*ps*) is positioned in front of or besides *pc*. The generation of daughter spaces has to ensure that *ps* lies completely in (the reference corner of) one of these spaces. The placement of *ps* is recorded for the respective daughter space. Thus, this daughter space already includes a placement when it enters the stack *SStack*.

The procedure offers some other features such as the merger of residual spaces that is aimed at larger and often better loadable residual spaces. A daughter space that has just been generated is merged where possible with a residual space that is already in the stack. Two residual spaces are only merged if a placing has not been reserved for either of them and the two residual spaces together form a rectangle. The merger of residual spaces as well as two other features of the procedure is parameterised by means of the so-called layer parameters which are explained in Table 3.

It can be stated that the GA will always generate layouts that satisfy the guillotine constraint. This is ensured on the one hand by the layer structure of layouts. On the other hand, each residual space within a layer, starting with the layer rectangle, is completely divided into two parts along a residual space dimension (i.e. parallel to a container dimension). A piece situated in the residual space is always placed completely on one or other of the two parts. The orientation constraint is met if necessary by avoiding non-permitted orientation variants of the pieces.

Table 3
Layer parameters

Parameter	Value	Explanation
rscut		Mode for generating daughter residual spaces
	0	Daughter spaces are generated such that the area of the larger daughter space is maximised
	1	Daughter spaces are generated such that the area of the smaller daughter space is maximised
ovmode		Mode for determining the orientation variants for the piece pair in a residual space
	0	Orientation variants are determined such that the x-dimension (in the direction of the layer depth) of the larger piece is as small as possible and (in case of tie) the x-dimension of the smaller piece is also as small as possible
	1	Orientation variants are determined such that the sum of the x-dimensions of both pieces is as large as possible
rsmerge		Mode for residual space merging
	0	Without residual space merging
	1	With residual space merging

2.6. Learning the layer parameters

Experiments with the different possible settings for the three layer parameters showed that none of the eight possible value combinations, when fixed, dominates the other seven combinations to a sufficient extent. In addition changing the value combinations during the search often proves advantageous. To cope with this situation a learning concept is tested for the layer parameters, which is arranged as follows:

- The values for the parameters *rscut*, *ovmode* and *rsmerge* are re-determined before the generation of each layer. The selection of the parameter values in each case is done randomly in accordance with a dynamic probability distribution for the eight possible value combinations.
- To generate the start generation a pre-defined start distribution is selected, which will be discussed later. The probability distribution is updated on each transition to a successor generation.
- The procedure is as follows: for each value combination i, a success frequency $succ\_frequ(i)$, $i = 1, \dots, 8$, is introduced and is initialised at the start of the search with zero. If a solution is taken into a new generation, $succ\_frequ(i)$ is increased by the value j, if j layers of the solution were generated in accordance with the value combination i. Caused by the merger mutation solutions may be removed from the new generation that have already been accepted. In this case the affected success frequencies have to be corrected downwards correspondingly. The relative success frequencies after n generations $(n = 1, 2, \dots)$ finally define the probability distribution of the value combinations used for the $(n+1)$th generation.

The dynamisation of the distribution is aimed at the preferred selection of the most successful value combinations in the previous course of the search, where it is assumed that these will continue to be particularly successful in the future.

3. Adaptation of the genetic algorithm to the strip packing problem

In the following the adaptation of the CLP-GA to the strip packing problem is explained. Furthermore additional modifications will be discussed that serve to improve the solution quality. Finally, the GA for the 2D-SPP has to be configured.

3.1. Taking account of the variable container length

The solution of the strip packing problem is led back to the calculation of multiple instances of the container loading problem with reducing container lengths. To this end, the CLP-GA is embedded into a control procedure that is shown in Fig. 7. The control procedure is as follows:

- Initially a start solution *sstart* for the given SPP instance is constructed by means of the heuristic BFDH* which is explained later. The best solution *sbest* is set to *sstart* and the minimal container length *lCbest* is initialised accordingly. The operator $lU(s)$ provides the length (x-dimension) used by a (complete or partial) solution s.

```
procedure sp_search(in: container width wC, set of pieces P,
                    out: best solution sbest, minimal container length lCbest)
calculate the start solution sstart by means of the heuristic BFDH*;
sbest := sstart; lCbest := lU(sstart);
if |P| > nplarge then
    P' := ∅; skept := sstart;
    for l := sstart.nl to 1 by -1 do
        P' :=P'∪E {p ∈ P | piece p is placed in layer l of sstart}
        skept := skept \ {layer l of sstart};
        if |P'| > npsmall then break; endif;
    endfor;
else P' := P; skept := ∅;
endif;
initialise container length lC := lCbest – 1 – lU(skept);

{lC-loop}
repeat
    {solve 2D-CLP instance for current length lC}
    cl_genetic_search(lC, wC, P', s');
    if solution s' includes all pieces p ∈ P' then
        sbest :=s'∪E skept; lCbest := lU(s') + lU(skept);
        lC := lU(s') – 1; {reduce lC for next cycle}
    endif;
until (s' does not include all pieces p ∈ P');
end.
```

Fig. 7. Procedure *sp_search*.

- The further proceeding depends on the instance size. If the number of pieces $|P|$ exceeds the limit *nplarge* only a subset $P' \subset P$ is used to calculate further solutions. In order to determine P' the layers of *sstart* are examined in ascending order according to the filling rate. For each layer the set P' is extended by all pieces placed in the layer. This process stops if the number of pieces in P' exceeds the limit *npsmall*. The remaining layers of *sstart* with the highest filling rates are maintained in the partial solution *skept*. Suitable values for the parameters *nplarge* and *npsmall* will be indicated later (cf. Table 4).
- If the number of pieces $|P|$ does not exceed *nplarge* the set P' is set to P and *skept* is set to empty.
- Now the current container length lC is initialised as the difference $lCbest - 1 - lU(skept)$; of course lU $(skept)$ is equal to zero if *skept* is empty.
- Each time the following (lC-)loop is passed through, another 2D-CLP instance with the container dimensions lC, wC and the piece set P' is solved by means of the CLP-GA.
- Granted that the returned solution s' includes all pieces in P'. Then the union of the new (partial) solution s' and the kept partial solution

Table 4
Standard configuration of the GA for the 2D-SPP

Parameter settings for the integrated CLP-GA

- Population size $npop = 50$
- No. of solutions to be reproduced per generation $nrep = 10$
- Use probabilities: crossover—$pcross = 0.67$, standard mutation—$pmut = 0.33$
- No. of merger mutations per generation $nmerge = 10$
- Percentages $qldp_i$, $i = 1,2,3$, (cf. Table 2) depend on the no. of piece types *nptypes* of the given CLP instance as follows:

nptypes lies in	[1, 49]	[41, 60]	[61, 200]	>200
$qldp_1$	100	66	10	5
$qldp_2$	100	66	10	5
$qldp_3$	33	33	33	33

- No. of calculated (subsequent) generations per CLP instance *ngen* depends on the no. of pieces *np* of the given instance as follows:

np lies in	[1, 60]	[61, 100]	>100
ngen	1000	500	100

Additional parameter settings

- No. of runs per SPP instance $nruns = 2$
- Start distributions for the probabilities of the value combinations of the layer parameters (cf. Table 3):

Value combination	1	2	3	4	5	6	7	8
rscut	0	0	0	0	1	1	1	1
ovmode	0	0	1	1	0	0	1	1
rsmerge	0	1	0	1	0	1	0	1
Probability								
1. run	0.5	0.5	0	0	0	0	0	0
2. run	0.2	0.2	0.2	0.2	0.05	0.05	0.05	0.05

- $nplarge = 199$, i.e. an instance is reduced for the CLP-GA if it contains at least 200 items; $npsmall = 100$, i.e. the no. of items in an reduced instance must exceed 100 (cf. Fig. 7) *skept* represents another solution to the SPP instance. Its total length amounts to $lU(s') + lU(skept)$ where $lU(s') \leqslant lC$. The container length lC is reduced at least by one unit before the next cycle begins. The initialisation and the updates of lC ensure that each new solution to the SPP instance is the best solution found so far.

- The search terminates the first time no solution s' including all pieces in P' was found. Moreover the search is aborted if the length *lCbest* reaches the well-known continuous lower bound *clb* that is calculated as

$$clb = \left\lceil \left(\sum_{p \in P} \text{area}(p) \right) / wC \right\rceil; \qquad (1)$$

$\lceil z \rceil$ stands for the smallest integer not smaller than z. The best solution *sbest* and the respective length *lCbest* are returned in either case.

The adaptation approach is supported by the circumstance that the CLP-GA terminates for any CLP instance just after a solution containing all delivered pieces was found. It turns out that new and better solutions to an SPP instance can be computed with low effort so long as the used container length is relatively far from the best possible value. Thus, the major part of the overall search effort accounts for a relatively small number of CLP instances.

For larger instances the heuristic BFDH* yields solutions of relatively high quality. Consequently, the start solution will already include a subset of layers with sufficiently high filling rates. Therefore, the further search can be focussed on the improvement of the poorer layers of the start solution. Through this the overall search effort may be considerably reduced.

3.2. Constructing the start solution

The heuristic BFDH* is derived from (the best variant of) the Best Fit Decreasing Height (BFDH) heuristic that was proposed by Mumford-Valenzuela et al. (2003). Subsequently the BFDH heuristic is described as a layer building method. Afterwards the modifications of the BFDH heuristic are introduced. As usual in the 2D-SPP literature, the dimensions of the container and of the pieces, respectively, are referred to as "width" and "height" (instead of "length") in this section.

The BFDH heuristic works as follows:

- At first the pieces are oriented in such a way that the width is greater than or equal to the height for each rectangle. After this the pieces are pre-sequenced by non-increasing height.
- Now the pieces are processed one after another. Each piece is packed into a rectangular layer, at the layer bottom and left justified. The width of a layer is given by the container width while the height of a layer is determined by the height of the first piece that is packed into the layer.
- If at least one (existing) layer can accommodate the current piece the layer with least remaining free (unoccupied) width is chosen in which the piece fits. Otherwise a new layer is opened above the existing layers and the current piece is stowed in the new layer as the first piece.

Two modifications of the BFDH heuristic are suggested here. The first modification deals with the selection of an existing layer for the current piece. As before the layer with minimal remaining free width is searched for that can accommodate the piece. But during this search the orientation of the piece is no longer fixed: For each layer both orientation variants of the piece are experimentally used. Thus, a successful search results in an existing layer and an orientation of the piece. Of course the current piece is then packed into the layer in the found orientation. If the search was not successful the piece is stowed into a new layer in the prescribed orientation "width $\geqslant$ height".

The second modification tries to utilise the remaining free space within layers above the pieces at the layer bottom. The modification is realised by means of an extra routine. Each time a piece has been packed into a layer it is examined whether the remaining free width of the layer became too small to accommodate a further piece at the bottom. If this is the case the routine is once applied to the layer. The routine is as follows:

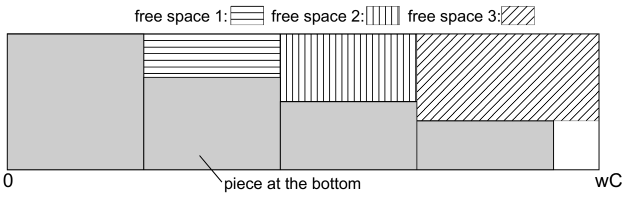


Fig. 8. Set-up of free spaces within a layer.

- First the pieces in the layer are sequenced (from the left to the right) by non-increasing height. Although this sequence is guaranteed by the original BFDH heuristic it is possibly disturbed by the first modification.
- Now several rectangular free spaces are generated within the layer as shown in Fig. 8. The free spaces are processed from the left to the right.
- For each free space the piece with the largest area is determined that fits into the space in any orientation. If a piece and an appropriate orientation were found the piece is fixed in the bottom-left corner of the free space. While the search for a fitting piece is successful it is repeated. After a piece was packed the free space is updated by shifting its left edge to the right edge of the packed piece. Thus the loading of a free space may result in several pieces which lie side by side within the free space.
- At last the layer arrangement is returned and the set of free pieces is updated where necessary. Consequently, those pieces which are packed by means of the routine are no more available for a packing at the bottom of a layer.

Obviously, the BFDH* heuristic satisfies the guillotine constraint. If the orientation constraint is to be met all steps are omitted or modified in a self-evident way in which pieces are rotated.

3.3. Diversifying the search

As explained above a learning process of the layer parameters (cf. Table 3) takes place within the CLP-GA. Whenever a CLP instance is solved these parameters are learned from scratch, i.e. they are selected first in accordance with a given start distribution.

Several suitable start distributions are defined using the relative strength of the individual parameters values. Because, for example, the setting $rscut = 0$ (cf. Table 3) usually achieves better results than the complementary value, the start distributions have correspondingly higher probabilities for the four value combinations where $rscut = 0$. Two start distributions are shown explicitly in Table 4.

Because different favourable start distributions exist, it appears reasonable to diversify the search for a solution of an SPP instance. The following diversification approach divides the search into several runs:

- With each run the complete search process is repeated for the SPP instance. For each CLP instance to be solved within a single run the same start distribution is used. But this distribution is changed on the transition to the next run.
- The procedure *sp_search* is called once for each run (cf. Fig. 7). However, from the second run onwards the best SPP solution found so far acts the role of the start solution which of course is not calculated again by the BFDH* heuristic. Thus the initial value of lC within a run is set taking account of the minimum used container length achieved in previous runs.
- The best SPP solution of all runs is updated after the end of each run where necessary and this solution is entered as the process solution at last.

3.4. Post-optimising the best solution of a run

The required layer structure of layouts prohibits pieces from penetrating layer borders. Thus, space may remain free on both sides of a layer which could actually be filled. The implied disadvantage of the layer approach can be tempered by an additional elementary heuristic for the post-optimisation of layouts. The idea of the heuristic consists of moving defined pieces in a layer into an adjacent layer and to use previously free space on the broad side of the adjacent layer for this purpose. This procedure is repeated for several layers. Where the procedure is successful, the used container length will be reduced by one or more units, in other words an improved solution of the strip packing problem is achieved.

The heuristic takes one layout as input and is divided into three phases, which are explained below:

- In the *analysis phase* the block structure is determined for each layer in addition to the layer structure of the layout (cf. Fig. 9a). Each layer is, in general, broken down into several rectangular blocks that follow one another along the broad side of the container (y-direction). One or more pieces are completely arranged in each block. The depth of a block results from the maximum sum of the depths (x-dimensions) of pieces in the block placed next to one another in the lengthways direction; the width of a block is defined analogously. The blocks in a layer are divided into critical and non-critical blocks. The depth (x-dimension) of the critical blocks is equal to the layer depth.
- The task of the *reorganisation phase* is to create a rectangular free space, called M space, at one of the broad sides of every single layer, which can then be used for displacing pieces from other layers (cf. Fig. 9b). The width dimension (y-dimension) of this space, which is designated the free width, should be as large as possible; the depth dimension must be at least one length unit. The free width of a layer is maximised mainly through a reorganisation of its blocks. These are rearranged (with the pieces they contain) in such a way that first the critical and then the non-critical blocks follow one another without any gaps. In addition, single pieces at the relevant broad side of the layer are displaced parallel to the layer width or stored elsewhere in previously free space on the inside of the layer. In addition, individual blocks are reflected at their centre line parallel to the x-axis.
- In the *displacement phase* a limited number of the layers with the maximum free widths are initially selected. The selected layers are subjected experimentally to a displacement process in every possible permutation or sequence in the container. The best layout with the shortest length over all permutations obtained by the displacement process is then completed by the remaining layers, which retain their original layer depths. This layout is returned as the solution for the heuristic.
- The *displacement process* for a given sequence of the selected layers is demonstrated in Figs. 10 and 11. Fig. 10 shows the selected layers with

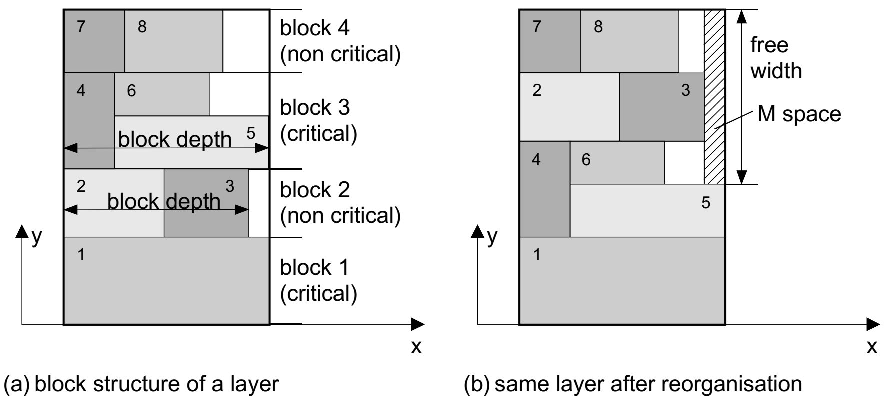


Fig. 9. Block structure and reorganisation of a layer.

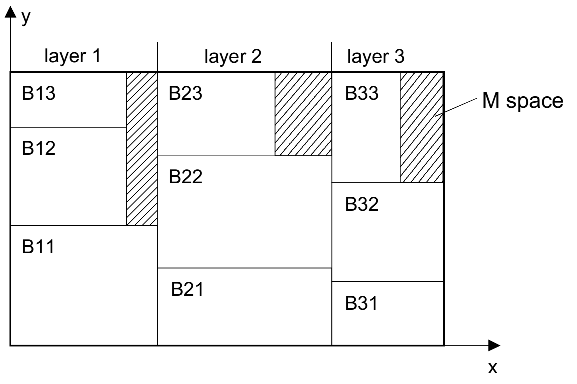


Fig. 10. Three layers with blocks B_{ij} $(i,j = 1,2,3)$ after the reorganisation phase.

their blocks after the reorganisation phase, where the *M* spaces are in an analogue situation (top right).

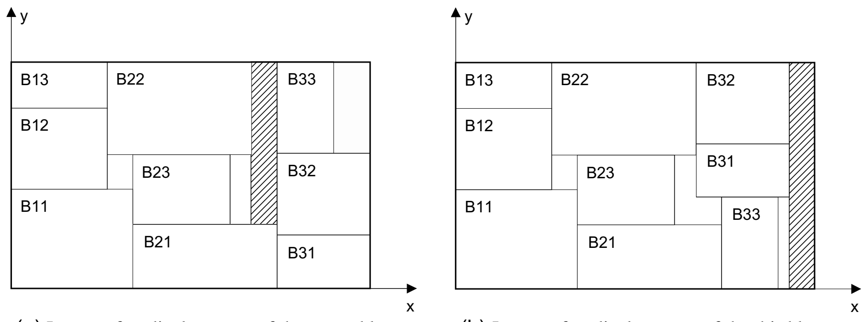
- Starting with the second layer of a permutation, in each case a subset of the critical blocks in the layer i $(i > 1)$ is determined in such a way that the sum of the block widths is the maximum of, but does not exceed, the free width of the layer $i - 1$. The blocks in layer i are now rearranged so that the selected critical blocks can be displaced by being pushed to the left into the *M* space of layer $i - 1$. Following this the (maximum) displacement is carried out. Fig. 11a shows the displacement process for the second layer. Before block *B22* can be pushed to the left, it has to be exchanged for block *B23*. After the displacement the *M* space of layer i is redefined taking account of the layer's old *M* space and the displacement. Because of the displacement, the width of the *M* space, i.e. the free width of layer i, always increases.
- Fig. 11b shows the displacement process for the third layer. After block *B33* has changed places with the critical blocks *B31* and *B32*, the latter can be pushed into the *M* space of the second layer. The width of the third layer is now equal to the container width. Its depth therefore indicates the achieved reduction of the container length that was used for the given permutation.

The heuristic generates solutions that, in general, no longer satisfy the guillotine constraint and it is therefore only applicable if this constraint is not present. This can be seen in Fig. 11b, where, for example, the pieces in block *B11* cannot be reproduced through guillotine cuts.

The post-optimising heuristic is incorporated into the method in the following way. At the end of each run (i.e. after the *lC*-loop was left, cf. Fig. 7) the heuristic is started and fed by the best solution found in the run. If the heuristic is able to improve this solution the best solution of the run (and the respective container length) is updated.

3.5. Configurating the genetic algorithm for the SPP

A standard configuration of the GA for the SPP was determined empirically. It contains all necessary parameter settings and is listed in Table 4 (see above). The standard configuration is used



(a) Layers after displacement of the second layer. (b) Layers after displacement of the third layer.

Fig. 11. Results of the displacement process.

throughout the numerical test described below. The variation of the values for the parameters $qldp_i$ ($i = 1, 2$) and *ngen* serves to control the trade off between solution quality and effort. The termination of the GA is ruled on the one hand by the respective specifications within the procedure *sp_search* (cf. Fig. 7) and by the number of runs *nruns*. On the other hand, a time limit is imposed on the entire search process that is spread evenly on the runs.

4. Procedure test

The GA, hereafter designated as SPGAL ("L" stands for "layer approach"), was implemented in C. In the following, a benchmark test is described that includes all methods which are listed in Table 1. The test was carried out on a Pentium PC with a core frequency of 2 GHz.

4.1. Sets of 2D-SPP instances

In order to consider all methods in Table 1 different sets of 2D-SPP instances from the literature are used which are listed in Table 5. Some of the instance sets are combined with different SPP subtypes (cf. Section 1) by disregarding the orientation constraint that is originally required or by adding the guillotine constraint. For each set of instances it is indicated which of the four subtypes (RF, RG, OF and OG) are taken into account.

For the instance sets 1, 4 and 7 optimal solutions of the instances are known, since for each instance a larger rectangle has been cut into smaller pieces without any waste. Additional data that concern single instances or subsets of instances are given in the following tables. For further information the reader is referred to the specified literature.

4.2. Numerical results

Each of the subsequent tables includes the results provided by the method SPGAL for one of the instance sets described above. Furthermore, available results of other methods from Table 1 are quoted. The following remarks concern the execution of the test as well as the set-up of the tables:

- If subtype RG or OG is required with an instance set then SPGAL is run without the post-optimisation heuristic to satisfy the guillotine constraint. Otherwise (subtypes RF, OF) the post-optimisation heuristic is used.
- For each instance set the results of SPGAL are indicated in the same way as the results of the comparison methods in the literature. Thus, the solution quality is measured in different ways. "lU" always refers to the used container length in length units. Optimum lengths and lower bounds are also given in length units. The (percentage) "filling rate" of a solution is computed as (total area of all pieces)/$(lU \cdot wC)$. Other quality measures are explained below if necessary.
- For the instance sets 1 to 6 ten calculations per instance were carried out. The data "best" and "mean", given for SPGAL, always refer to the best and mean result, respectively, of the ten calculations. For the instance set 7 from Mumford-Valenzuela et al. (2003) each instance was computed once only.
- A time limit of 1200 seconds per instance was imposed throughout. The mean calculation time per instance in Pentium-2 GHz seconds required by SPGAL is always denoted as "mean time". Calculation times of other methods are not quoted since they are often not available.

4.2.1. Results for the instances from Jakobs

The results for the instances from Jakobs (1996) are shown in Table 6. For each of the methods the considered subtype is indicated. The method SPGAL provides the best overall result although it was tested–just as the GA from Mumford-Valenzuela et al. (2003)–for the more difficult subtype RG.

4.2.2. Results for the instances from Dagli et al.

The results for the instances from Ratanapan and Dagli (1997, 1998) and Dagli and Poshyanonda (1997) can be found in Table 7. Each of the methods was tested for the subtype RF. For each

Table 5
Sets of 2D-SPP instances

No.	Authors, source	No. of instances	Considered subtypes	Further comments
1	Jakobs (1996)	2	RF, RG	• Optimum lengths are known for variant RF
2	Ratanapan and Dagli (1997, 1998) and Dagli and Poshyanonda (1997)	4	RF	• In short "instances from Dagli et al."
3	Kröger (1993, 1995)	12	RF, RG	
4	Hopper and Turton (2000)	21	RF, OF, OG	• Optimum lengths are known for variant RF • Set consists of seven classes C1–C7 each with three instances; the instances of a class match in terms of container width, optimum length (RF) and no. of pieces
5	Berkey and Wang (1987) and Iori et al. (2002)	300	OF	• Instances were originally proposed for the 2D bin packing problem and adapted to the 2D-SPP by disregarding the bin sizes • Set consists of six classes C1–C6 each with 50 instances; each class is defined by a specific interval for the piece dimensions • Each class consists of five subclasses with 10 instances each; the instances of a subclass match in terms of container width and no. of pieces
6	Martello and Vigo (1998) and Iori et al. (2002)	200	OF	• See comments on the instances from Berkey and Wang • Set consists of four classes C1–C4
7	Mumford-Valenzuela et al. (2003)	480	RG	• Optimum lengths are known • Set consists of two classes "Nice" and "Path(ological)"; pieces of an instance of class Nice are similar in shape and size; pieces of an instance of class Path vary more extreme in shape and size • Both classes consist of eight subclasses with identical numbers of pieces per instance

Table 6
Results for the instances from Jakobs (1996), subtypes RF and RG

Instance	No. of pieces	Optimum length; width *wC*	Jakobs (1996)	Liu and Teng (1999)	Mumford-Valenzuela et al. (2003)	SPGAL	
			Subtype RF lU	Subtype RF lU	Subtype RG lU	Subtype RG lU	
			Best	Best	Best	Best	Mean
1	25	15; 40	17	16	16	16	16.0
2	50	15; 40	17	16	16	15	15.0
Mean time							13

Table 7
Results for the instances from Dagli et al. (cf. Table 5), subtype RF

Instance	No. of pieces	Continuous lower bound; width *wC*	Ratanapan and Dagli (1997, 1998)	Dagli and Poshyanonda (1997)	Poshyanonda and Dagli (2004)	SPGAL	
			Filling rate (%)	Filling rate (%)	Filling rate (%)	Filling rate (%)	
			Best	Best	Best	Best	Mean
1	31	45; 60	91.88	–	–	95.67	93.9
2	21	40; 60	92.50	–	–	97.56	96.6
3	37	112; 30	94.41	–	96.03	98.58	98.5
4	37	161; 20	–	97.15	–	97.62	97.6
Mean time							21

of the instances the SPGAL performs considerably better than the comparison methods.

4.2.3. Results for the instances from Kröger

In Table 8 the results for the instances from Kröger (1993) are listed. Both comparison procedures generate layouts that satisfy the guillotine constraint. The procedure SPGAL was tested for each of the subtypes RG and RF.

The parallel GA (PGA) of Schnecke is clearly dominated by both of the other methods. If the guillotine constraint is considered (RG), the best values of Kröger's PGA and of the SPGAL coincide, on average, where the comparison procedure is better for smaller instances, SPGAL for larger instances. Measured over all calculations, even without subsequent optimising, SPGAL achieves an improvement of 0.5 length units per instance compared to Kröger's PGA.

If the guillotine constraint is omitted (RF), the SPGAL achieves new best values for five instances and equals the best values of Kröger's PGA for the remaining instances. The length reduction determined over all instances and calculations is now

Table 8
Results for the instances from Kröger (1993)[a], subtypes RF and RG

Instance	No. of pieces	Continuous lower bound	Schnecke (1996)		Kröger (1993)		SPGAL			
			Subtype RG lU		Subtype RG lU		Subtype RG lU		Subtype RF lU	
			Best[b]	Mean[b]	Best[c]	Mean[c]	Best	Mean	Best	Mean
1	25	107	111	111.8	109	109.4	110	110.9	109	109.8
2	25	103	107	107.6	104	105.0	105	105.2	104	105.0
3	25	102	105	106.8	104	104.2	105	105.2	104	104.4
4	35	151	156	158.3	152	153.0	153	153.0	151	152.6
5	35	122	127	127.5	123	123.4	124	124.0	123	123.7
6	35	123	127	128.4	124	124.6	125	125.4	124	124.9
7	45	194	202	202.8	196	197.0	196	196.1	195	195.5
8	45	163	171	172.8	164	165.2	164	164.9	164	164.5
9	45	133	139	139.2	134	135.2	134	135.0	134	134.9
10	60	249	256	259.6	253	253.8	251	251.5	250	251.0
11	60	275	286	288.0	279	280.8	277	277.2	276	276.6
12	60	280	292	294.6	283	284.6	281	281.9	280	281.4
Mean 1–12	–	166.8	173.3	174.8	168.7	169.7	168.7	169.2	167.8	168.7
Mean time								156		166

[a] For each instance the container width is $wC = 100$.
[b] Best/mean value of at least 10 calculations.
[c] Best/mean value of five calculations.

1.0 units. SPGAL misses the lower bounds by a maximum of two units, in the average of the instances by a single unit only, and generates (proven) optimum solutions for two instances.

4.2.4. Results for the instances from Hopper and Turton

The results for the instances from Hopper and Turton (2000) and for the subtype RF are presented in Table 9. The quality of a solution is measured by the percentage gap, i.e. the relative distance to the (known) optimum length $lOpt$. The gap is computed as $(lU - lOpt)/lOpt$. The indicated gaps for the seven classes are averaged over the respective three instances.

SPGAL clearly dominates the three comparison procedures from Hopper and Turton (2000). Measured over all classes, the gaps achieved for the best solutions are at least 3.4 percentage points below the corresponding values in these comparison procedures; for the mean values from all calculations the distance to the best solutions of the comparison procedures is still at least 3.0 percentage points, on average, over all classes.

SPGAL generates optimal solutions for 15 of the 21 instances; for the remaining six instances (2, 7, 8, 16, 19 and 21) the optimum is missed in each case by a single length unit.

In Table 10 the results for the instances from Hopper and Turton (2000) and for the subtypes

Table 9
Results for the instances from Hopper and Turton (2000), subtype RF

Class	No. of pieces	Optimum length $lOpt$; width wC	Hopper and Turton (2000)			SPGAL	
			GA + BLF Gap (%)	NE + BLF Gap (%)	SA + BLF Gap (%)	Gap (%)	
			Best	Best	Best	Best	Mean
C1	16–17	20; 20	4.0	5.0	4.0	1.7	1.7
C2	25	15; 40	7.0	7.0	6.0	0.0	0.9
C3	28–29	30; 60	5.0	4.0	5.0	2.2	2.2
C4	49	60; 60	3.0	4.0	3.0	0.0	1.4
C5	72–73	90; 60	4.0	4.0	3.0	0.0	0.0
C6	97	120; 80	4.0	4.0	3.0	0.3	0.7
C7	196–197	240; 160	5.0	5.0	4.0	0.3	0.5
Mean C1–C7	–	–	4.6	4.7	4.0	0.6	1.0
Mean time							139

Table 10
Results for the instances from Hopper and Turton (2000), subtypes OF and OG

Class	Continuous lower bound clb	Iori et al. (2002)	SPGAL			
		Subtype OF Gap (%)	Subtype OF Gap (%)		Subtype OG Gap (%)	
		Best	Best	Mean	Best	Mean
C1	20	1.59	1.59	1.59	3.17	3.17
C2	15	2.08	2.08	3.33	2.08	3.33
C3	30	2.15	3.16	3.16	3.16	3.92
C4	60	4.75	2.70	3.52	2.70	3.78
C5	90	3.92	1.46	2.03	1.46	2.38
C6	120	4.00	1.64	1.72	1.64	1.88
C7	240	–	1.23	1.52	1.64	1.69
Mean C1–C6	–	3.08	2.10	2.56	2.37	3.08
Mean C1–C7	–	–	1.98	2.41	2.26	2.88
Mean time				159		143

OF and OG are provided. Now the method SPGAL is compared to the best method from Iori et al. (2002) (cf. Table 1). The percentage gap of a solution is computed as $(lU - clb)/lU$ where clb stands for the continuous lower bound. Again, the gaps of the classes are averaged over the respective instances.

For the lower classes C1–C3 with smaller instances both methods achieve the same solution quality or the comparison method is superior. For the higher classes including larger instances SPGAL performs considerably better than the comparison method. A better overall result is also obtained if the more challenging subtype OG is required for SPGAL.

4.2.5. Results for the instances from Berkey and Wang and from Martello and Vigo

Tables 11 and 12 present the results obtained for the instances from Berkey and Wang (1987) and for the instances from Martello and Vigo (1998), respectively. For both instance sets the subtype OF is assumed. Again SPGAL is compared to the best method proposed by Iori et al. (2002). The gap of a solution now represents the relative distance to an advanced lower bound, denoted here as alb,which bases on the relaxation of the 2D-SPP to the so-called One-dimensional Contiguous Bin Packing Problem (cf. Iori et al., 2002). The percentage gap is computed as $(lU - alb)/lU$ and the gap of each subclass as well as the lower bound alb is averaged over the respective ten instances.

The method SPGAL yields the better overall result for the instance set from Berkey and Wang as well as for the instance set from Martello and Vigo. Together these instance sets consist of 10 classes and 50 subclasses. Averaged over the respective instances SPGAL performs better for seven classes while the method from Iori et al. (2002) is superior for three classes. SPGAL achieves smaller averaged gaps for 34 subclasses whereas the comparison method performs better for 12 subclasses. Both methods reach the same gap for four subclasses. If only the 30 subclasses are considered the instances of which include at least 60 items, the respective figures are 24, 3 and 3.

4.2.6. Results for the instances from Mumford-Valenzuela et al.

In Table 13 the results for the instances from Mumford-Valenzuela et al. (2003) and for the subtype RG are listed. Mumford-Valenzuela et al. (2003) compared their GA to some simple and fast heuristics such as the Best Fit Decreasing Height (BFDH) heuristic described above. The best results which were achieved by these heuristics are indicated in column 4 while column 5 includes the results of the GA developed by Mumford-Valenzuela et al. Apart from the subclass Nice-4 the best performing heuristic is always BFDH. In column 6 the results of the BFDH* heuristic are presented, i.e. the quality of the start solution of the SPGAL method is measured. In column 7 the results of the complete SPGAL method are shown while column 8 includes the CPU times consumed by the complete SPGAL method.

On the one hand the BFDH* heuristic performs constantly better than each of the (simple) heuristics which were tested in Mumford-Valenzuela et al. (2003). Furthermore the BFDH* heuristic never needs more than a few seconds. On the other hand the complete SPGAL method achieves the best result for each of the classes and subclasses, respectively. On average, the improvements are larger for the class "Path". For both classes, the improvements decrease with increasing size of the instances.

The numerical test can be summarised as follows:

- The test was conducted using very different instances from the literature with regard to the problem size, the SPP subtype, the container width and other characteristics.
- For each instance set and for each SPP subtype the method SPGAL achieves the best overall result and the computing times remain reasonable in either case.
- Some of the competing methods which were examined, e.g., the parallel GA from Kröger (1993), are competitive or perform even better than SPGAL for smaller instances with less than 50 items. However, for larger instances with 50 items or above, SPGAL proves to be the dominant method. Clearly, the superiority

Table 11
Results for the instances from Berkey and Wang (1987), subtype OF

Class/subclass	No. of pieces	Width *wC*	Lower bound *alb*	Iori et al. (2002)	SPGAL		
				Gap (%)	Gap (%)	*lU*	
				Best	Best	Best	Mean
C11	20	10	60.3	1.33	2.15	61.60	61.96
C12	40	10	121.6	0.30	0.36	122.00	122.30
C13	60	10	187.4	0.86	0.83	189.00	189.15
C14	80	10	262.2	0.23	0.23	262.80	262.90
C15	100	10	304.4	0.37	0.19	305.00	305.25
Mean C1	–	–	187.2	0.62	0.75	188.08	188.31
C21	20	30	19.7	0.99	3.91	20.50	20.53
C22	40	30	39.1	2.19	0.00	39.10	39.54
C23	60	30	60.1	2.44	0.00	60.10	60.53
C24	80	30	83.2	1.76	0.11	83.30	83.36
C25	100	30	100.5	1.27	0.20	100.70	100.73
Mean C2	–	–	64.5	1.93	0.84	60.74	60.94
C31	20	40	157.4	4.42	5.72	166.70	167.34
C32	40	40	328.8	3.08	2.08	335.40	336.87
C33	60	40	500.0	3.48	2.02	509.80	511.10
C34	80	40	701.7	2.49	1.53	712.50	714.50
C35	100	40	832.7	1.94	1.23	842.60	844.42
Mean C3	–	–	504.1	3.08	2.52	513.40	514.85
C41	20	100	61.4	6.33	7.53	66.30	66.72
C42	40	100	123.9	5.79	2.58	127.10	127.95
C43	60	100	193.0	4.53	1.83	196.60	197.65
C44	80	100	267.2	4.19	1.84	272.20	273.58
C45	100	100	322.0	3.14	1.62	327.30	328.56
Mean C4	–	–	193.5	4.80	3.08	197.90	198.89
C51	20	100	512.2	4.51	4.42	536.60	537.80
C52	40	100	1053.8	2.95	2.60	1081.40	1082.75
C53	60	100	1614.0	3.32	2.34	1650.80	1654.09
C54	80	100	2268.4	1.73	1.38	2299.50	2302.99
C55	100	100	2617.4	3.07	1.93	2666.90	2672.55
Mean C5	–	–	1613.2	3.12	2.53	1646.98	1650.04
C61	20	300	159.9	8.56	10.85	179.10	179.77
C62	40	300	323.5	6.52	4.10	337.00	338.89
C63	60	300	505.1	5.04	2.84	519.80	522.87
C64	80	300	699.7	4.43	2.74	719.40	724.18
C65	100	300	843.8	3.57	2.81	868.10	872.41
Mean C6	–	–	506.4	5.62	4.67	524.68	527.62
Mean C1–C6	–	–	–	3.20	2.40	521.96	523.44
Mean time							96

for larger instances is of greater importance since smaller instances may be also calculated by an exact method (cf., e.g., Martello et al., 2003).

Table 12
Results for the instances from Martello and Vigo (1998), subtype OF

Class/subclass	No. of pieces	Width *wC*	Lower bound *alb*	Iori et al. (2002)	SPGAL		
				Gap (%)	Gap (%)	*lU*	
				Best	Best	Best	Mean
C11	20	100	490.4	2.45	2.62	502.70	503.15
C12	40	100	1049.7	1.03	0.99	1059.40	1060.79
C13	60	100	1515.9	0.89	0.90	1529.70	1530.56
C14	80	100	2206.1	0.82	0.76	2222.90	2225.93
C15	100	100	2627.0	0.73	0.81	2648.80	2650.22
Mean C1	–	–	1577.8	1.18	1.22	1592.70	1594.13
C21	20	100	434.6	7.66	6.60	465.90	467.45
C22	40	100	922.0	5.85	3.58	956.20	962.00
C23	60	100	1360.9	5.27	2.73	1398.90	1407.01
C24	80	100	1909.3	5.23	2.93	1967.30	1976.13
C25	100	100	2362.8	4.85	2.44	2422.30	2432.41
Mean C2	–	–	1397.9	5.77	3.66	1442.12	1449.00
C31	20	100	1106.8	0.00	0.00	1106.80	1107.01
C32	40	100	2189.2	0.07	0.11	2191.20	2191.71
C33	60	100	3410.4	0.00	0.19	3417.50	3417.51
C34	80	100	4578.6	0.20	0.20	4588.10	4588.15
C35	100	100	5430.5	0.08	0.08	5434.90	5434.92
Mean C3	–	–	3343.1	0.07	0.12	3347.70	3347.86
C41	20	100	337.8	4.88	5.11	354.20	355.31
C42	40	100	642.8	4.65	3.29	664.70	666.76
C43	60	100	911.1	4.48	2.28	932.60	935.29
C44	80	100	1177.6	4.61	2.43	1207.40	1211.17
C45	100	100	1476.5	4.29	2.09	1507.80	1512.37
Mean C4	–	–	909.2	4.58	3.04	933.34	936.18
Mean C1–C4	–	–	–	2.90	2.01	1828.97	1831.79
Mean time							95

5. Summary

In this paper a GA for the two-dimensional strip packing problem with rectangular pieces was proposed. Each of the four subtypes of the SPP which result if the orientation constraint and the guillotine constraint are optionally considered is covered by the GA.

The GA generates layer-type structured layouts. It was acquired through adaptation of a genetic algorithm for the container loading problem. Tracing the solution of the SPP to the solution of several CLP instances with reducing container lengths proved to be a suitable adaptation approach. Further components of the GA are the BFDH* heuristic that provides the start solution for an SPP instance, the diversification of the search with regard to the learning process of layer parameters and a heuristic for the post-optimising of SPP solutions. For large instances the search is focussed on the improvement of the poorer layers of the start solution.

The GA was subjected to a comprehensive numerical test including more than 1000 benchmark instances from the literature with up to 5000 items. A representative sample of eleven recently developed 2D-SPP methods which base on very different design principles (cf. Table 1) was

Table 13
Results for the instances from Mumford-Valenzuela et al. (2003)[a], subtype RG

Class/subclass	No. of instances	No. of pieces	Mumford-Valenzuela et al. (2003)		SPGAL		
			Heuristic[b]	GA[b]	BFDH*[b]	GA[b]	Mean time[c]
Nice-1	50	25	118.9	107.3	116.2	105.2	103
Nice-2	50	50	115.5	107.8	113.3	105.0	542
Nice-3	50	100	110.7	108.6	109.1	105.3	365
Nice-4	50	200	108.2	111.3	107.2	105.5	216
Nice-5	10	500	105.4	120.8	104.6	103.7	496
Nice-6	10	1000	104.2	–	103.4	102.7	504
Nice-7	10	2000	103.0	–	102.4	102.0	323
Nice-8	10	5000	101.9	–	101.7	101.5	305
Path-1	50	25	121.1	104.4	113.6	102.8	59
Path-2	50	50	120.3	108.5	114.1	102.6	327
Path-3	50	100	118.0	112.6	112.0	103.1	381
Path-4	50	200	115.2	116.7	110.2	106.7	50
Path-5	10	500	110.6	120.8	107.5	105.4	93
Path-6	10	1000	109.4	–	107.4	104.6	111
Path-7	10	2000	107.2	–	105.1	104.0	195
Path-8	10	5000	105.1	–	–[d]	–[d]	–

[a] Container width wC and optimum length amount to 100 length units for each instance.
[b] lU averaged over the instances of a subclass.
[c] Mean CPU time in 2 GHz seconds per instance.
[d] Not calculated since the respective instances could not be coded by means of the data type int (4 Bytes).

considered for comparison purposes. Compared to these methods the GA provided the better overall result in either case. For larger instances with at least 50 items the GA proved to be the clearly dominant method.

Finally, the features of the GA which are mainly responsible for the solution quality are stressed:

- The search is carried out by means of problem-specific operators directly in the space of completely defined layouts with a layer structure. In accordance with the classification of solution approaches for the SPP from Hopper and Turton (2000), the GA is classified in group 3.
- If solutions are generated for a given CLP instance some random based decisions are made in order to ensure a sufficient flexibility of the search. But in view of the placing of pieces, these random based decisions only provide a framework determining, e.g., the supply of free pieces.
- In fact, pieces are placed and sequences of new layers are generated by means of sophisticated and expensive greedy procedures that consider the mutual relations of the pieces. For example, pieces are not packed independently of each other since in general two pieces are loaded simultaneously into a residual space. Furthermore, the overall quality of a sequence of new layers is taken into account. In this way the GA takes a more global view on the problem data as a whole.

In contrast to the GA proposed here most of the comparison methods include simpler placing heuristics which regard the mutual relations of the pieces only to a smaller extent. Consequently, generated layouts are based on random to a stronger degree. This applies in particular for some methods in group 1 (cf. Table 1) in which solutions are encoded as placing sequences and the pieces are processed only separately. Therefore, the superiority of the GA especially for larger and complexer instances appears plausible.

Up to now the placing of pieces is tailored to the guillotine constraint; hence, an extension of the GA by subtype-specific placing routines could lead to a further improvement of the solution quality. This remains a topic of future research.

Acknowledgment

The author wishes to thank the three anonymous referees for their constructive comments.

References

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

Bortfeldt, A., Gehring, H., 1999. Two metaheuristics for strip packing problems. In: Despotis, D.K., Zopounidis, C., (Eds), Proceedings of the Fifth International Conference of the Decision Sciences Institute, Athens, vol. 2, pp. 1153–1156.

Bortfeldt, A., Gehring, H., 2001. A hybrid genetic algorithm for the container loading problem. European Journal of Operational Research 131, 143–161.

Coffman Jr., E.G., Shor, P.W., 1990. Average-case analysis of cutting and packing in two dimensions. European Journal of Operational Research 44, 134–144.

Dagli, C.H., Poshyanonda, P., 1997. New approaches to nesting rectangular patterns. Journal of Intelligent Manufacturing 8, 177–190.

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

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

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

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

Iori, M., Martello, S., Monaci, M., 2002. Metaheuristic algorithms for the strip packing problem. In: Pardalos, P., Korotkich, V. (Eds.), Optimization and Industry: New Frontiers. Kluwer Academic Publishers.

Jakobs, S., 1996. On genetic algorithms for the packing of polygons. European Journal of Operational Research 88, 165–181.

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

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

Liu, D., Teng, H., 1999. An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles. European Journal of Operational Research 112, 413–420.

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

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

Martello, S., Pisinger, D., Vigo, D., 1998. The three-dimensional bin packing problem. Operations Research 48, 256–267.

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

Monaci, M., 2001. Algorithms for Packing and Scheduling Problems. Ph.D. Thesis, Università degli studi di Bologna.

Mumford-Valenzuela, C.L., Vick, J., Wang, P.Y., 2003. Heuristics for large strip packing problems with guillotine patterns: An empirical study. In: Metaheuristics: Computer Decision-Making. Kluwer Academic Publishers B.V., pp. 501–522.

Poshyanonda, P., Dagli, C.H., 2004. Genetic neuro-nester. Journal of Intelligent Manufacturing 15, 201–218.

Ratanapan, K., Dagli, C.H., 1997. An object-based evolutionary algorithm for solving rectangular piece nesting problems. In: IEEE (Eds.), Proceedings of the IEEE Conference on Evolutionary Computation 1997, ICEC '97, IEEE, Piscataway, NJ, USA, pp. 989–994.

Ratanapan, K., Dagli, C.H., 1998. An object-based evolutionary algorithm: The nesting solution. In: IEEE (Eds.), Proceedings of the International Conference on Evolutionary Computation 1998, ICEC '98, IEEE, Piscataway, NJ, USA, pp. 581–586.

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

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