European Journal of Operational Research 215 (2011) 57–69

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

A two-stage intelligent search algorithm for the two-dimensional strip packing problem

Stephen C.H. Leung [a], Defu Zhang [b,*], Kwang Mong Sim [c]

[a] Department of Management Sciences, City University of Hong Kong, Hong Kong
[b] Department of Computer Science, Xiamen University, Xiamen 361005, China
[c] Department of Information and Communications, Gwangju Institute of Science and Technology, South Korea

ARTICLE INFO

Article history:
Received 16 May 2010
Accepted 2 June 2011
Available online 13 June 2011

Keywords:
Packing problem
Heuristic search
Simulated annealing

ABSTRACT

This paper presents a two-stage intelligent search algorithm for a two-dimensional strip packing problem without guillotine constraint. In the first stage, a heuristic algorithm is proposed, which is based on a simple scoring rule that selects one rectangle from all rectangles to be packed, for a given space. In the second stage, a local search and a simulated annealing algorithm are combined to improve solutions of the problem. In particular, a multi-start strategy is designed to enhance the search capability of the simulated annealing algorithm. Extensive computational experiments on a wide range of benchmark problems from zero-waste to non-zero-waste instances are implemented. Computational results obtained in less than 60 seconds of computation time show that the proposed algorithm outperforms the supposedly excellent algorithms reported recently, on average. It performs particularly better for large instances.

1. Introduction

Strip packing problem belongs to a well-known family of combinatorial optimization problems and has many industrial applications in different fields, such as cutting rectangular pieces from a large sheet of material in wood and glass industries, placing goods on shelves in the warehousing industry, arranging articles and advertisements on pages in newspapers, and so on. In addition, this problem can be extended to the container loading problem also. For more extensive literature on cutting and packing problems, interested readers may refer to Dowsland (1992), Lodi et al. (2002) and Wäscher et al. (2007).

In this paper, a two-dimensional (2D) orthogonal strip packing problem (SPP) is considered. This problem can be stated as follows. Given a rectangular board of given width W and unlimited length, and a set of n rectangles, each rectangle i ($1 \leqslant i \leqslant n$) with length l_i and width w_i, the orthogonal SPP is to place n rectangles on the board such that no two rectangles overlap and the used board length h is minimized. This paper assumes that edges of rectangles are parallel to edges of the rectangular board, i.e. all rectangles are packed orthogonally. A classification of the 2D Bin Packing Problem that can also be applied to the 2D SPP and 2D Knapsack Problem was introduced by Lodi et al. (1999). According to this classification, there are four subtypes of the SPP problem:

- RF: the pieces may be rotated 90° (R) and no guillotine cutting is required (F);
- RG: the pieces may be rotated 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).

This paper mainly addresses the subtype OF, the types OG, RG are not considered at all.

The 2D SPP belongs to a subset of classical cutting and packing problems (Wäscher et al., 2007) and has been shown to be NP-hard (Hochbaum and Maass, 1985). Some exact algorithms for the orthogonal two-dimensional cutting and packing problem have been proposed in the past by Beasley (1985a), Christofides and Hadjiconstantinou (1995), Martello et al. (2003), Hifi and M'Hallah (2005), Fekete et al. (2007), Cui et al. (2008) and Kenmochi et al. (2009). However, they might not be practical for large scale problems because a large amount of running time is required to obtain optimal solutions. Therefore, constructive heuristic algorithms, which can quickly produce good approximation solutions, are preferred for solving this class of problems. Some examples are the well-known bottom-left (BL) and bottom-left-fill (BLF) (Baker et al., 1980; Chazelle, 1983; Berkey and Wang, 1987), best fit heuristics (Burke et al., 2004), recursive heuristics (Zhang et al., 2006), bricklaying heuristics (Zhang et al., 2008), and new heuristics (Ortmann et al., 2010). These heuristics may be grouped into

* Corresponding author. Tel.: +86 592 5918207; fax: +86 592 2580258.
E-mail address: dfzhang@xmu.edu.cn (D. Zhang).

0377-2217/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2011.06.002

classes of level algorithms, shelf algorithms, plane algorithms and other algorithms (Ortmann et al., 2010).

Metaheuristic methods such as genetic algorithms (Jakobs, 1996; Ramesh, 1999; Dowsland, 2006; Gonçalves, 2007), simulated annealing algorithms (Dagli and Hajakbari, 1990; Lai and Chan, 1996), neural network algorithms (Dagli and Poshyanonda, 1997), and some hybrid metaheuristic algorithms (Hopper and Turton, 2001; Hifi and M'Hallah, 2003; Lesh et al., 2005) are presented and surveyed. These algorithms can produce good solutions within acceptable time, say, less than half-an-hour. However, generally speaking, these heuristic algorithms are still time-consuming and are less practical for larger problems. It is of great interest to note that BF + metaheuristic is developed by adding a metaheuristic phase that first uses BF to pack some rectangles and then applies BLF + metaheuristic for the remaining rectangles (Burke et al., 2009). A genetic algorithm called SPGAL, which works directly on the solution layouts and does not involve encoding of solutions, is presented by Bortfeld (2006). A very effective heuristic algorithm to solve the rectangle packing problem, based on the corner-occupying action and caving degree, was presented by Huang et al. (2007). A greedy randomized adaptive search procedure (GRASP) that involves learning some benchmark instances to determine the desirable parameter settings for the strip packing problem has also been presented (Alvarez-Valdes et al., 2008). Algorithms based on metaheuristics can find excellent solutions within acceptable time. Belov et al. (2008) developed two algorithms, SVC and BS, based on one-dimensional heuristics. SVC is a very powerful algorithm and is even better than GRASP, on average. Recently, a least waste first heuristic was presented, which evaluates positions used by rectangles (Wei et al., 2009). BF + SA is described as one of the best algorithms for strip packing problem with rotation constraints (Burke et al., 2009). GRASP and SVC are viewed as excellent among algorithms developed for strip packing problems with and without rotation constraints. Heuristic algorithms are very simple and fast while metaheuristics are time-consuming but they (metaheuristics) can generally find better solutions than heuristic algorithms. In this paper, an intelligent search algorithm based on a heuristic algorithm, local search and simulated annealing algorithm is presented to solve the orthogonal strip packing problem where the heuristic algorithm is very simple and efficient. That guarantees simulated annealing algorithm has more time to search better solutions. Computational results of a wide range of benchmark problems have shown that the presented heuristic outperforms algorithms described as excellent in the literature, on average. In particular, our algorithm performs better for large instances.

The rest of this paper is organized as follows. In Section 2, based on a simple scoring rule, a new heuristic algorithm is presented to derive the sequence for packing a set of rectangles. Section 3 presents a two-stage intelligent search algorithm based on a local search and a simulated annealing algorithm. Section 4 studies the selection of some strategies and parameters to find an efficient algorithm. Computational results of a wide range of zero-waste and non-zero-waste instances are reported in Section 5. Conclusions are summarized in Section 6.

2. Constructive heuristic algorithm

The constructive heuristic algorithm is suitable for quickly obtaining a solution, within an acceptable time. Our heuristic algorithm scores each unpacked rectangle for a given current available space *s*. This space *s* can be determined by a structure that includes five variables (Fig. 1(1)), namely, the position of the space (the most left denoted as *x*, and the lowest as *y*), width of space *w*, height of the left wall h_1, and height of the right wall h_2.

There are six available spaces in Fig. 1(1), their respective positions marked by numbers 1–6. In order to save these six spaces, we use a structure array $S[m]$, where $m = 6$, in Fig. 1(1). Each available space is, in turn, saved in $S[1]$, $S[2]$, $S[3]$, $S[4]$, $S[5]$, and $S[6]$ by increasing ordering of *x* -coordinate of the lower left corner of the space. For example, space *s* in Fig. 1(1) is determined by $S[4].x$, $S[4].y$, $S[4].w$, $S[4].h_1$ and $S[4].h_2$. This structure is very convenient to determine the lowest and the most left space, and to update space *S*. For example, one rectangle is packed into space *s*, as in Fig. 1(2), and then we need to insert one space into the structure array, when space *S* is updated as in Fig. 1(2). The number of available spaces *m* increases by 1. *m* is changed dynamically and remains less than *n*, which can help determine the lowest and the most left space more rapidly.

For a given rectangle *i* and space *s*, scoring rules (for calculating fitness value *fitness*) of rectangle *i* are as shown in Table 1.

Where $r[i].width$ and $r[i].length$ denote width and length, respectively, of rectangle *i*, and *fitness* denotes the score of unpacked rectangle *i* for space *s* which can be computed according to one of the two cases. The intuitive meaning of the case where $h_1 \geqslant h_2$ can be observed from Fig. 1; for example, $fitness \leftarrow 4$ in the 3rd line of score (i, h_1, h_2, w) corresponds to Fig. 1(4), which shows that the left wall is higher than the right wall. The rectangle

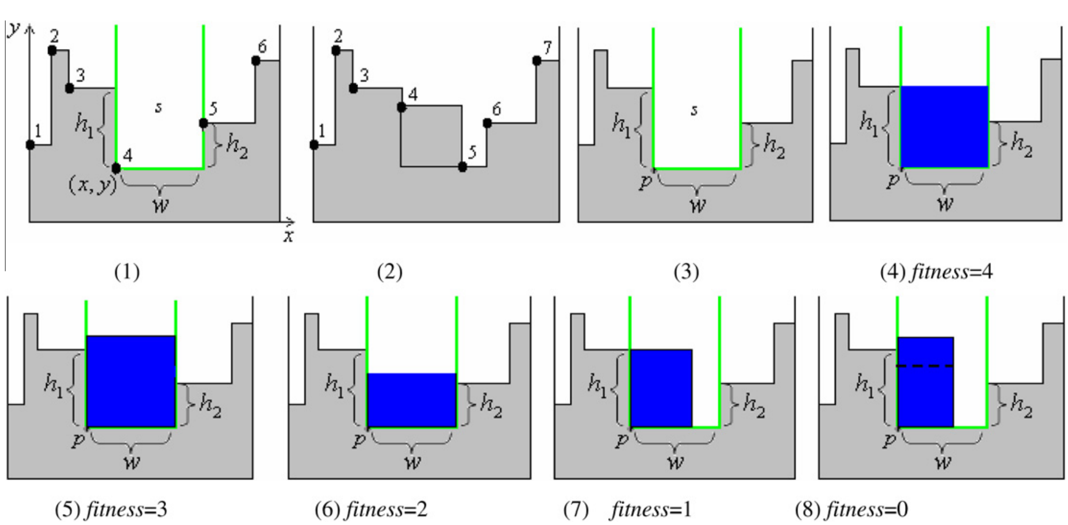

Fig. 1. Available space and computation of the fitness value.

Table 1
Scoring function score (i, h_1, h_2, w) for calculating fitness value *fitness* and the change of number of spaces *m*.

Conditions		*fitness*	*m*	Conditions		*fitness*	*m*
$h_1 \geqslant h_2$	$w = r[i].width$ and $h_1 = r[i].length$	4	-2 or -1	$h_1 < h_2$	$w = r[i].width$ and $h_2 = r[i].length$	4	-1
	$w = r[i].width$ and $h_1 < r[i].length$	3	0		$w = r[i].width$ and $h_2 < r[i].length$	3	0
	$w = r[i].width$ and $h_1 > r[i].length$	2	0		$w = r[i].width$ and $h_2 > r[i].length$	2	0
	$w > r[i].width$ and $h_1 = r[i].length$	1	0		$w > r[i].width$ and $h_2 = r[i].length$	1	0
	$w > r[i].width$ and $h_1 \neq r[i].length$	0	+1		$w > r[i].width$ and $h_1 \neq r[i].length$	0	+1

that meets this case should be assigned the maximum score. This rectangle can fit three edges and its top edge fits the top edge of the right wall, so this case is assigned a higher fitness value, namely, *fitness* = 4. *fitness* ← 3 in line five corresponds to Fig. 1(5). From Fig. 1, we can observe that the score for each case almost corresponds to the practical packing case. Similarly, the corresponding figure can be given for the case where $h_1 < h_2$.

The space array *S* is updated after one rectangle is packed; number of spaces *m* can increase by one (+1), or it can decrease by one or two (−1 or −2), or it may remain unchanged (0). The details are summarized in Table 1, where *m* will decrease by two (−2) for $h_1 = h_2$ *and fitness* = 4. In addition, the number of spaces *m* can decrease by more than one when the width of one space *s* is less than the minimum width of unpacked rectangles (*minwidth*), which will lead to merging of the space. For example, spaces *S*[5] and *S*[6] will be merged into *S*[5] if *S*[5].w < *minwidth,* and at the same time, *S*[5].*y* ← S[6].*y*, *S*[6] ← *S*[7], *m* ← *m* − 1.

The constructive heuristic algorithm can be designed in detail as shown in Fig. 2.

Where *h* denotes the length of the used board, *pin* denotes the number of packed rectangles, and *minwidth* denotes the minimum width of unpacked rectangles. HeuristicPacking(*X*) works as follows. The **while** loop of lines 2–18 iterates as long as there are some unpacked rectangles. Line 3 determines the lowest and the most left space *s*. Line 4 judges whether there exists one rectangle that can be packed into *s*; if it is true, it shows the number of unpacked rectangles to be packed into *s* is at least 1. Otherwise, line 18 is executed, and space array *S* is updated. Each unpacked rectangle is scored for the current available space *s* in lines 5–6. If rectangle *R* can be packed into *s*, *pin* and *h* are updated in lines 8 and 9, respectively. Whether rectangle *R* is packed near to the left wall or the right wall is determined by $S[s].h_1 \geqslant S[s].h_2$. *S* will be changed after rectangle *R* is packed into *s*, so *S* will be updated.

The presented heuristic considers two available positions of one available space *s*, then selects one position according to the size of *S*[*s*].*h*1 and *S*[*s*].h_2. However, most popular heuristics only consider the left and the bottommost position, such as the heuristics in Babu and Babu (1999) and Hifi and M'Hallah (2003). In particular, one novel scoring rule proposed is to select one rectangle for a given position.

3. Two-stage intelligent search algorithm

3.1. Local search

For a given sequence *X* of rectangles, we can obtain a solution by HeuristicPacking(*X*) mentioned above. It is noted that performance of the heuristic algorithm significantly depends on packing sequence *X* of rectangles. Some research results have shown that the packing sequence of rectangles affects the performance of the presented algorithms (Alvarez-Valdes et al., 2008). Therefore, we can make use of local search to improve the solution of the heuristic algorithm. Since performance of local search is significantly affected by the quality of initial packing sequence, in this paper, a packing sequence sorted by non-increasing ordering of perimeter size is selected as the initial packing sequence; the reason is explained in the next section. According to this packing sequence, large rectangles are first considered for packing. In fact, packing rectangles strictly according to their perimeter ordering does not correspond to practical packing cases in industry. Therefore, we can use local search to try different orderings to find a better solution. The process of local search (LS) is shown in Fig. 3

Where best*h* denotes the best length of all packing orderings tried up to now, and current*h* is the length obtained by HeuristicPacking(*Х́*) for ordering *Х́*. LS() first sorts unpacked rectangles by non-increasing ordering of perimeter size, then HeuristicPacking(*X*) is called in line 2 and we obtain an initial best*h*. Then, LS executes the two for loops in lines 3 and 4. For given *i* and *j*, the algorithm first swaps the order of rectangles *i* and *j* in the current ordering *X* and obtains a new ordering *Х́*, and then computes current*h* in the new ordering *Х́*. If current*h* is less than best*h*, then it

```
HeuristicPacking(X)
1   h ←0; pin←0;
2   while pin<n do
3       find the lowest and the most left space s;
4       if S[s].w≥minwidth then
5           for each unpacked rectangle i do
6               s_i =score(i, S[s].h1, S[s].h2, S[s].w);
7           select one rectangle R with the maximum score from unpacked rectangles;
8           pin←pin+1;
9           if S[s].y+r[R].length>h then
10              h←S[s].y+ r[R].length;
11          if S[s].h1≥S[s].h2 then
12              pack rectangle R near to the left wall;
13              update space array S;
14          else
15              pack rectangle R near to the right wall;
16              update space array S;
17      else
18          update spaces array S;
19  return h;
```

Fig. 2. Constructive heuristic algorithm.

```
LS()
1  sort all unpacked rectangles by non-increasing ordering of perimeter size and obtain ordering X;
2  besth←HeuristicPacking(X), bestX←X;
3  for i←1 to n-1 do
4      for j←i+1 to n do
5          swap the order of rectangles i and j in X and obtain a new ordering X´;
6          currenth←HeuristicPacking(X´);
7          if currenth<besth then
8              besth←currenth;
9              bestX←X´, X←X´;
10 return besth and bestX;
```

Fig. 3. Local search algorithm.

updates best*h* and best*X*. This process is repeated until execution of the two for loops is finished. LS() makes use of the greedy idea during the process of search because the current solution will be improved while one neighborhood of the current solution is better than the current solution. HeuristicPacking(X') is a subprocess; LS() will be more efficient if HeuristicPacking(X') can return a solution fast and efficiently.

3.2. Simulated annealing algorithm

When LS() stops, it can find a good solution, but the solution cannot be improved further. Simulated annealing is a powerful random-based metaheuristic algorithm and is used in this paper to improve the solution of LS().

Since simulated annealing is a randomized local exploration optimization heuristic algorithm, it can be seen as an improvement of local search by allowing some controlled uphill movements so that it obtains the global optimal solution. Simulated annealing accepts a better solution absolutely. Accepting a move toward a worse solution depends on the current temperature, and on the gap between objective function values for the two solutions. Temperature (T) is a key parameter used to control the process of search.

Since simulated annealing has powerful capability of escaping the local optimal trap, it has been applied to solve some packing problems (Dagli and Hajakbari, 1990; Lai and Chan, 1996; Burke et al. 2009). In order to solve the strip packing problem, the standard simulated annealing can be modified as in Fig. 4.

Where the initial packing ordering *X* comes from LS(); it inherits the good structure of the solution obtained by LS(). The stop criterion in this paper is as follows. Running time is not larger than 60 seconds or when the lower bound is found. Function rand(0, 1) returns a real number between 0 and 1, *exp*(x) is a math function, namely, e^x, where e is Euler's number. SA() accepts the better ordering in line 8 and updates best*h*, SA() accepts the worse ordering with probability as in line 12, where the probability depends on T and best*h*- current*h*. The cooling operations adopt geometrical cooling in line 13 after each of the loops is finished, where α is the cooling rate.

3.3. Multi-start strategy

Since SA() inherits packing orderings obtained by LS, it is very difficult to change the structure of packing orderings to escape from the local optimal trap. The multi-start strategy encourages the search process to examine unvisited solution regions. The process of the multi-start strategy is shown in Fig. 5.

Were line 3 and line 5 generate different packing orderings so that the search process can visit different solution regions. Experimental test in Section 4 shows that the multi-start strategy is very efficient.

3.4. Two-stage intelligent search algorithm

According to LS and SA, the two-stage intelligent search algorithm is as stated in Fig. 6.

Where Multi-start() is added after line 13 in SA() which makes SA a multi-start local search. The computational results show that multi-start local search is more successful than a pure SA strategy. LS algorithm is a deterministic algorithm, while SA() is a randomized algorithm. Since time limit of $\leqslant 60$ seconds is imposed in this paper, we must consider the stop criterion for enforcing the time limit in LS(), and then ISA() may stop during the process of executing LS() for large instances ($n \geqslant 1000$). Therefore, SA() will have no effect on solutions of large instances, though it can still improve the quality of solutions for small instances.

4. Effect of sorting strategy, parameters and multi-start strategy

Some research results have shown that initial ordering has great effect on the performance of algorithms (Alvarez-Valdes et al., 2008). In particular, the performance of LS depends on selection

```
SA()
1  give an initial temperature T0 , T ← T0;
2  while the stop criterion is not yet satisfied do
3      for i ← 1 to L do
4          randomly select two rectangles j and k in X;
5          obtain a new ordering X´ by swapping the order of rectangles j and k;
6          currenth←HeuristicPacking(X´);
7          if currenth<besth then
8              besth←currenth;
9              bestX←X´, X←X´;
10         else
11             if exp((besth- currenth)/T)≥(rand(0,1)) then
12                 X←X´;
13     T←αT;
14 return besth and bestX;
```

Fig. 4. Simulated annealing algorithm.

```
Multi-start()
randomly flip a coin;
if coin comes up heads then
    sort all the rectangles by non-increasing ordering of perimeter size and obtain ordering X;
else
    sort all the rectangles by non-increasing ordering of width size and obtain ordering X;
return X;
```

Fig. 5. Multi-start strategy.

```
ISA()
LS();
SA();
return besth;
```

Fig. 6. Two-stage intelligent search algorithm.

of the initial solution. In order to choose the better sorting strategy for LS algorithm, we have done a computational study using LS() for the following data set:

ZDF:16 problem instances generated by combining non-zero-waste instances in 2sp with zero-waste instances N and CX (Leung and Zhang, 2011), where 2sp includes 38 problem instances which are gcut (Beasley, 1985b), ngcut (Beasley, 1985a), cgcut (Christofides and Whitlock, 1977) and Beng (Bengtsson, 1982).

The reason we select it is that in most instances ISA() will exit before SA() is run, SA() has almost no effect on improvement of the solution. The computational results are reported in Fig. 7.

From Fig. 7, we can observe that sorting by perimeter, area, width and length can yield the total *Gap* values of 45.8, 48.2, 46.2 and 67.8, respectively, for data set ZDF. Sorting by perimeter can obtain the best result, while sorting by length can obtain the worst result. So the sorting strategy selected by us is sorting by perimeter, and it is still the reason why we select the sorting strategy by perimeter and width in Multi-start().

For SA(), settings of some parameters have great effect on the performance of simulated annealing algorithm; we have reported computational results of different parameter settings. We first select $T_0 = 0.5$, then we study the effect of parameters α in SA(). A non-zero-waste data set gcut with 13 instances (Beasley, 1985b) is selected for testing. Characteristics of this data set are that it is very difficult to obtain their optimal solutions for some instances,

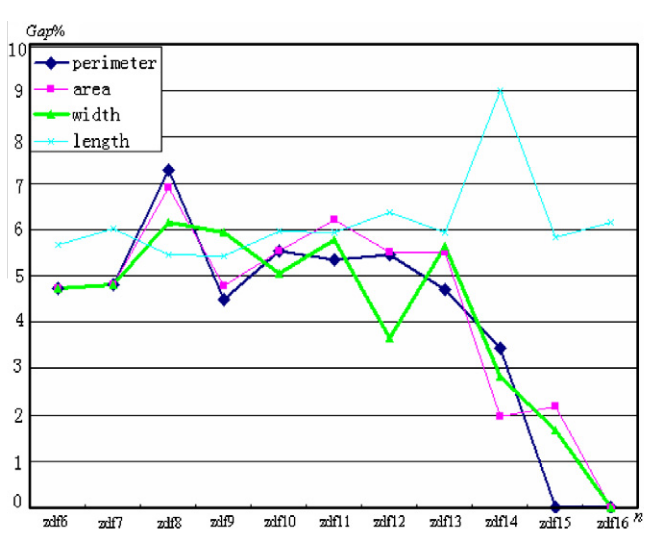

Fig. 7. Effect of sorting strategy.

and the size of the problem is small ($n \leqslant 50$). When α is 0.9, 0.91, 0.92, 0.93, 0.94, 0.95, the total *Gap* is 77.9, 77.8, 78, 77.7, 77.8, and 77.7, respectively. So we select $T_0 = 0.5$ and $\alpha = 0.93$ for experiments in the next section.

In addition, we investigated the performance of the presented algorithm with and without the multi-start strategy, i.e. the effect of multi-start strategy. We have done a computational study on a non-zero-waste data set Nice from Nice1 to Nice6 (Valenzuela and Wang, 2001). The reason we selected this data set is that these instances are more difficult and the problem size changes from 25 to 1000. The computational results are reported in Fig. 8. From Fig. 3, we can observe that the presented algorithm with multi-start strategy can obtain better results than that without multi-start strategy. The presented algorithm with multi-start strategy can significantly improve the results when problem instances solved are of small size. This algorithm yields almost no improvement while $n > 500$. The reason is that the presented algorithm is allowed to run for only 60 seconds.

5. Computational results

In order to verify the performance of ISA, we compare it with algorithms which are currently believed to be excellent, by testing it on a wide range of benchmark problems from extant literature.

Benchmark instances used by this paper are as follows.

- C: 21 problem instances generated by Hopper and Turton (2001);
- N: 13 problem instances generated by Burke et al. (2004);
- NT: 70 problem instances generated by Hopper (2001), of which the first 35 correspond to guillotine patterns, while the later 35 correspond to non-guillotine patterns;
- CX: 7 problem instances generated by Pinto and Oliveira (2005);
- BWMV: 500 problem instances of which instances from C01 to C06 are generated by Berkey and Wang (1987), and C07–C10 are generated by Martello and Vigo (1998); and
- Nice and Path: 72 problem instances generated by Valenzuela and Wang (2001); they are the floating-point data sets of both similarly dimensioned rectangles (Nice1–Nice5t) and vastly differing dimensions (Path1–Path5t).

All instances mentioned above are available at http://59.77.16.8/Download.aspx#p4. C, N, NT and CX are zero-waste instances, and their optimal solutions are known. In particular, CX is an extra large data set. 2sp, BWMV, Nice and Path, and ZDF are non-zero-waste instances, where optimal solutions involve some wasted regions also. Optimal solutions of some non-zero-waste instances are known, others are not known, but we can calculate their lower bounds (LB). In addition, one simple instance, Babu, generated by Ramesh (1999) is included.

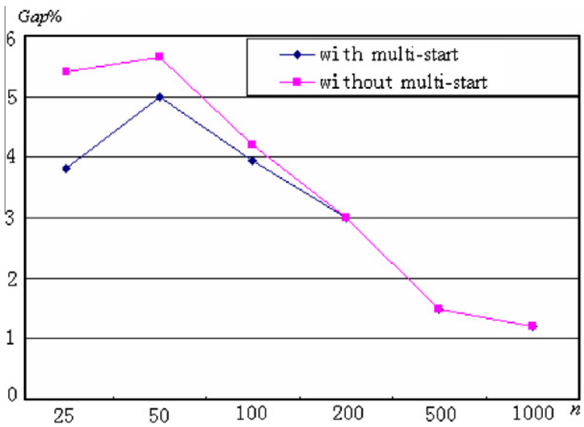

Fig. 8. Effect of multi-start strategy.

BF + SA (BSA) (Burke et al., 2009), GRASP (Alvarez-Valdes et al., 2008) and SVC (Belov et al., 2008) are among the supposedly excellent algorithms in the current literature, selected for comparison with ISA. ISA is coded in C++, GRASP executable programme is obtained from Prof. Ramon Alvarez-Valdes, and SVC executable programme is obtained from Dr. G. Belov. The three algorithms were run on a machine with Intel(R) Xeon(R) CPU E5405 2.00 GHz 1.99 GB RAM, and each run was allowed a maximum of 60 seconds duration. BSA is the combination of Best-fit and metaheuristics, and when run 10 times with a time limit of 60 seconds per run, the *best* solution is reported. Computational results of BSA are taken from Burke et al. (2009). It is noted that BSA allows the rectangles to rotate, considering RF subtype, while GRASP, SVC and ISA do not, considering only OF subtype. Computational results are reported in Tables 2–10. The symbols in the tables have the following meanings: *Instance* denotes problem instance, *n* is the number of rectangles, *W* is the width of the rectangular board, *LB* is the lower bound of the objective value of the problem instance, and LB is the optimal height for zero-waste problem. For non-zero-waste problem, LB in Tables 6 and 7 is from Iori et al. (2003), LB in Table 8 and 9 is calculated by $\lceil \sum_{i=1}^{n} l_i w_i / W \rceil$. *best* and *mean* denote the *best* and *mean* results, respectively, obtained by BSA, GRASP, SVC and ISA, over 10 runs. It is noted that *best* and *mean* are the same for SVC. *Gap%* is the percent gap to the lower bound (*LB*), namely, $Gap = 100 \times (mean - LB)/LB$. Calculation of *Gap* for BSA needs to replace *mean* with *best*. *Ave.* is the average of numerical values in each

Table 2
Computational results on data set C.

Instance				BSA	SVC	GRASP		ISA		Gap%			
	n	*W*	*LB*	Best	Mean	Best	Mean	best	Mean	BSA	SVC	GRASP	ISA
C11	16	20	20	20	20	20	20	20	20.0	0.0	0.0	0.0	0.0
C12	17	20	20	**20**	21	20	**20**	20	**20.0**	0.0	5.0	0.0	0.0
C13	16	20	20	20	20	20	20	20	20.0	0.0	0.0	0.0	0.0
C21	25	40	15	16	**15**	15	**15**	15	**15.0**	6.7	0.0	0.0	0.0
C22	25	40	15	16	**15**	15	**15**	15	**15.0**	6.7	0.0	0.0	0.0
C23	25	40	15	16	**15**	15	**15**	15	**15.0**	6.7	0.0	0.0	0.0
C31	28	60	30	31	**30**	30	**30**	30	**30.0**	3.3	0.0	0.0	0.0
C32	29	60	30	31	31	31	31	31	31.0	3.3	3.3	3.3	3.3
C33	28	60	30	31	**30**	30	**30**	30	**30.0**	3.3	0.0	0.0	0.0
C41	49	60	60	61	61	61	61	61	61.0	1.7	1.7	1.7	1.7
C42	49	60	60	61	61	61	61	61	61.0	1.7	1.7	1.7	1.7
C43	49	60	60	61	61	61	61	60	**60.9**	1.7	1.7	1.7	1.5
C51	73	60	90	91	91	91	91	91	91.0	1.1	1.1	1.1	1.1
C52	73	60	90	91	91	91	91	90	**90.8**	1.1	1.1	1.1	0.9
C53	73	60	90	92	**91**	91	**91**	91	**91.0**	2.2	1.1	1.1	1.1
C61	97	80	120	122	**121**	122	122	121	**121.0**	1.7	0.8	1.7	0.8
C62	97	80	120	121	121	121	121	121	121.0	0.8	0.8	0.8	0.8
C63	97	80	120	122	**121**	122	122	121	**121.0**	1.7	0.8	1.7	0.8
C71	196	160	240	244	**242**	244	244	242	**242.0**	1.7	0.8	1.7	0.8
C72	197	160	240	244	242	243	243	241	**241.0**	1.7	0.8	1.3	0.4
C73	196	160	240	245	**242**	243	243	242	**242.0**	2.1	0.8	1.3	0.8
Ave.				83.62	82.95	83.19	83.19	82.76	**82.84**	2.33	1.03	0.95	**0.76**

Table 3
Computational results of the data set N.

Instance				BSA	SVC	GRASP		ISA		Gap%			
	n	*W*	*LB*	Best	Mean	Best	Mean	Best	Mean	BSA	SVC	GRASP	ISA
N1	10	40	40	40	40	40	40	40	40.0	0.0	0.0	0.0	0.0
N2	20	30	50	50	50	50	50	50	50.0	0.0	0.0	0.0	0.0
N3	30	30	50	51	**50**	51	51	50	50.1	2.0	0.0	2.0	0.2
N4	40	80	80	82	81	81	81	80	**80.0**	2.5	1.3	1.3	0.0
N5	50	100	100	103	**101**	102	102	101	**101.0**	3.0	1.0	2.0	1.0
N6	60	50	100	102	101	101	101	100	**100.9**	2.0	1.0	1.0	0.9
N7	70	80	100	104	101	101	101	100	**100.0**	4.0	1.0	1.0	0.0
N8	80	100	80	82	81	81	81	81	81.0	2.5	1.3	1.3	1.3
N9	100	50	150	152	151	151	151	150	**150.9**	1.3	0.7	0.7	0.6
N10	200	70	150	152	151	151	151	150	**150.8**	1.3	0.7	0.7	0.5
N11	300	70	150	153	151	151	151	150	**150.7**	2.0	0.7	0.7	0.5
N12	500	100	300	306	**301**	304	304	301	**301.0**	2.0	0.3	1.3	0.3
N13	3152	640	960	964	963	965	965	960	**960.0**	0.4	0.3	0.5	0.0
Ave.				180.08	178.62	179.15	179.15	177.92	**178.18**	1.78	0.63	0.95	**0.41**

Table 4
Computational results of the large data set NT.

Instance				SVC	GRASP		ISA		Gap%		
	n	*W*	LB	Mean	Best	Mean	Best	Mean	SVC	GRASP	ISA
n1a	17	200	200	202	200	**200**	200	**200.0**	1.0	0.0	0.0
n1b	17	200	200	**200**	209	209	210	211.2	0.0	4.5	5.6
n1c	17	200	200	200	200	200	200	200.0	0.0	0.0	0.0
n1d	17	200	200	200	200	200	200	200.0	0.0	0.0	0.0
n1e	17	200	200	200	200	200	200	200.0	0.0	0.0	0.0
n2a	25	200	200	205	206	206	204	**204.0**	2.5	3.0	2.0
n2b	25	200	200	209	206	**206**	209	209.4	4.5	3.0	4.7
n2c	25	200	200	209	208	**208**	207	208.5	4.5	4.0	4.3
n2d	25	200	200	**207**	209	209	206	207.8	3.5	4.5	3.9
n2e	25	200	200	**205**	206	206	206	206.7	2.5	3.0	3.3
n3a	29	200	200	208	209	209	206	**206.1**	4.0	4.5	3.1
n3b	29	200	200	**207**	208	208	209	209.0	3.5	4.0	4.5
n3c	29	200	200	207	205	**205**	205	206.1	3.5	2.5	3.1
n3d	29	200	200	208	207	207	204	**204.3**	4.0	3.5	2.2
n3e	29	200	200	**207**	207	**207**	208	208.0	3.5	3.5	4.0
n4a	49	200	200	**205**	206	206	206	206.0	2.5	3.0	3.0
n4b	49	200	200	**205**	207	207	205	**205.0**	2.5	3.5	2.5
n4c	49	200	200	**205**	205	**205**	205	206.0	2.5	2.5	3.0
n4d	49	200	200	205	206	206	204	**204.8**	2.5	3.0	2.4
n4e	49	200	200	**205**	205	**205**	206	206.0	2.5	2.5	3.0
n5a	73	200	200	**204**	205	205	205	205.1	2.0	2.5	2.6
n5b	73	200	200	204	204	204	202	**203.6**	2.0	2.0	1.8
n5c	73	200	200	**204**	206	206	204	204.4	2.0	3.0	2.2
n5d	73	200	200	205	204	**204**	205	205.0	2.5	2.0	2.5
n5e	73	200	200	205	206	206	203	**204.7**	2.5	3.0	2.3
n6a	97	200	200	203	204	204	202	**202.8**	1.5	2.0	1.4
n6b	97	200	200	204	204	204	203	**203.0**	2.0	2.0	1.5
n6c	97	200	200	204	204	204	203	**203.6**	2.0	2.0	1.8
n6d	97	200	200	**202**	204	204.1	203	203.8	1.0	2.1	1.9
n6e	97	200	200	**203**	204	204	203	203.5	1.5	2.0	1.8
n7a	199	200	200	202	202	202	201	**201.0**	1.0	1.0	0.5
n7b	199	200	200	**202**	203	203	202	**202.0**	1.0	1.5	1.0
n7c	199	200	200	202	203	203	201	**201.9**	1.0	1.5	1.0
n7d	199	200	200	202	203	203	201	**201.9**	1.0	1.5	1.0
n7e	199	200	200	202	203	203	201	**201.9**	1.0	1.5	1.0
t1a	17	200	200	200	200	200	200	200.0	0.0	0.0	0.0
t1b	17	200	200	211	200	**200**	200	**200.0**	5.5	0.0	0.0
t1c	17	200	200	210	200	**200**	200	**200.0**	5.0	0.0	0.0
t1d	17	200	200	**200**	200	**200**	210	211.8	0.0	0.0	5.9
t1e	17	200	200	209	200	**200**	200	**200.0**	4.5	0.0	0.0
t2a	25	200	200	207	204	**204**	207	207.0	3.5	2.0	3.5
t2b	25	200	200	**205**	208	208	206	207.0	2.5	4.0	3.5
t2c	25	200	200	**206**	208	208	206	**206.0**	3.0	4.0	3.0
t2d	25	200	200	207	206	**206**	203	209.3	3.5	3.0	4.7
t2e	25	200	200	207	206	**206**	206	207.4	3.5	3.0	3.7
t3a	29	200	200	208	207	**207**	209	209.0	4.0	3.5	4.5
t3b	29	200	200	**207**	209	209	208	208.1	3.5	4.5	4.1
t3c	29	200	200	207	206	**206**	206	206.6	3.5	3.0	3.3
t3d	29	200	200	208	207	207	206	**206.4**	4.0	3.5	3.2
t3e	29	200	200	206	208	208	205	**205.0**	3.0	4.0	2.5
t4a	49	200	200	205	205	205	205	205.0	2.5	2.5	2.5
t4b	49	200	200	**205**	205	**205**	206	206.1	2.5	2.5	3.1
t4c	49	200	200	205	206	206	204	**204.9**	2.5	3.0	2.5
t4d	49	200	200	**205**	206	206	205	205.7	2.5	3.0	2.8
t4e	49	200	200	**205**	207	207	204	205.2	2.5	3.5	2.6
t5a	73	200	200	**204**	206	206	204	204.4	2.0	3.0	2.2
t5b	73	200	200	204	204	204	204	204.0	2.0	2.0	2.0
t5c	73	200	200	**204**	205	205	205	205.0	2.0	2.5	2.5
t5d	73	200	200	205	204	**204**	204	204.9	2.5	2.0	2.5
t5e	73	200	200	204	204	204	204	204.0	2.0	2.0	2.0
t6a	97	200	200	204	204	204	202	**203.2**	2.0	2.0	1.6
t6b	97	200	200	**202**	204	204	202	203.4	1.0	2.0	1.7
t6c	97	200	200	204	204	204	203	**203.0**	2.0	2.0	1.5
t6d	97	200	200	204	204	204	203	**203.5**	2.0	2.0	1.8
t6e	97	200	200	204	205	205	203	**203.5**	2.0	2.5	1.8
t7a	199	200	200	**201**	203	203	201	201.2	0.5	1.5	0.6
t7b	199	200	200	202	203	203	201	**201.0**	1.0	1.5	0.5
t7c	199	200	200	202	204	204	201	**201.0**	1.0	2.0	0.5
t7d	199	200	200	202	202	202	202	202.0	1.0	1.0	1.0
t7e	199	200	200	202	203	203	201	**201.7**	1.0	1.5	0.8
Ave.				204.54	204.64	204.64	203.93	**204.48**	2.27	2.32	**2.24**

Table 5
Computational results on the extra large data set CX.

Instance				SVC	GRASP		ISA		Gap%		
	n	*W*	*LB*	Mean	Best	Mean	Best	Mean	SVC	GRASP	ISA
50cx	50	400	600	**603**	613	613	613	620.2	0.5	2.2	3.4
100cx	100	400	600	616	617	617	614	**615.8**	2.7	2.8	2.6
500cx	500	400	600	604	605	605	601	**601.0**	0.7	0.8	0.2
1000cx	1000	400	600	601	602	602	600	**600.0**	0.2	0.3	0.0
5000cx	5000	400	600	600	600	600	600	600.0	0.0	0.0	0.0
10,000cx	10000	400	600	600	600	600	600	600.0	0.0	0.0	0.0
15,000cx	15000	400	600	600	600	600	600	600.0	0.0	0.0	0.0
Ave.				**603.43**	605.29	605.29	604.00	605.29	**0.57**	0.88	0.88

Table 6
Computational results on the non-zero-waste instances 2sp.

Instance				SVC	GRASP		ISA		Gap%		
	n	*W*	LB	Mean	Best	Mean	Best	Mean	SVC	GRASP	ISA
cgcut1	16	10	23	23	23	23	23	23.0	0.0	0.0	0.0
cgcut2	23	70	63	65	65	65	65	65.0	3.2	3.2	3.2
cgcut3	62	70	636	661	661	661	658	**660.2**	3.9	3.9	3.8
gcut1	10	250	1016	1016	1016	1016	1016	1016.0	0.0	0.0	0.0
gcut2	20	250	1133	**1187**	1191	1191	1187	**1187.0**	4.8	5.1	4.8
gcut3	30	250	1803	1803	1803	1803	1803	1803.0	0.0	0.0	0.0
gcut4	50	250	2934	3017	3002	**3002**	3003	3010.5	2.8	2.3	2.6
gcut5	10	500	1172	1273	1273	1273	1273	1273.0	8.6	8.6	8.6
gcut6	20	500	2514	2632	2627	**2627**	2632	2632.0	4.7	4.5	4.7
gcut7	30	500	4641	4693	4693	4693	4693	4693.0	1.1	1.1	1.1
gcut8	50	500	5703	**5876**	5912	5912	5883	5890.4	3.0	3.7	3.3
gcut9	10	1000	2022	2317	2317	2317	2317	2317.0	14.6	14.6	14.6
gcut10	20	1000	5356	5973	5964	**5964**	5964	5964.8	11.5	11.4	11.4
gcut11	30	1000	6537	6891	6899	6899	6869	**6884.4**	5.4	5.5	5.3
gcut12	50	1000	12,522	14,690	14,690	14,690	14,690	14690.0	17.3	17.3	17.3
gcut13	32	3000	4772	4977	4994	4994	4963	**4965.9**	4.3	4.7	4.1
ngcut1	10	10	23	23	23	23	23	23.0	0.0	0.0	0.0
ngcut2	17	10	30	**30**	30	**30**	31	31.0	0.0	0.0	3.3
ngcut3	21	10	28	28	28	28	28	28.0	0.0	0.0	0.0
ngcut4	7	10	20	20	20	20	20	20.0	0.0	0.0	0.0
ngcut5	14	10	36	36	36	36	36	36.0	0.0	0.0	0.0
ngcut6	15	10	29	31	31	31	31	31.0	6.9	6.9	6.9
ngcut7	8	20	20	20	20	20	20	20.0	0.0	0.0	0.0
ngcut8	13	20	32	34	33	**33**	34	34.0	6.3	3.1	6.3
ngcut9	18	20	49	51	50	**50**	52	52.0	4.1	2.0	6.1
ngcut10	13	30	80	80	80	80	80	80.0	0.0	0.0	0.0
ngcut11	15	30	50	52	52	52	52	52.0	4.0	4.0	4.0
ngcut12	22	30	87	87	87	87	87	87.0	0.0	0.0	0.0
beng1	20	25	30	**30**	30	**30**	31	31.0	0.0	0.0	3.3
beng2	40	25	57	57	57	57	57	57.0	0.0	0.0	0.0
beng3	60	25	84	84	84	84	84	84.0	0.0	0.0	0.0
beng4	80	25	107	107	107	107	107	107.0	0.0	0.0	0.0
beng5	100	25	134	134	134	134	134	134.0	0.0	0.0	0.0
beng6	40	40	36	36	36	36	36	36.0	0.0	0.0	0.0
beng7	80	40	67	67	67	67	67	67.0	0.0	0.0	0.0
beng8	120	40	101	101	101	101	101	101.0	0.0	0.0	0.0
beng9	160	40	126	126	126	126	126	126.0	0.0	0.0	0.0
beng10	200	40	156	156	156	156	156	156.0	0.0	0.0	0.0
Ave.				1539.05	1539.95	1539.95	1537.68	**1538.64**	2.80	**2.68**	3.02

column. Values shown in bold letters in Tables 2–10 denote the best results that BSA, GRASP, SVC and ISA can obtain for each instance.

5.1. Computational results of zero-waste problem instances

Table 2 shows results of zero-waste data sets C. For C, many among existing literature have reported computational results of their algorithms. For example, BLF + metaheuristics (SA, TS and GA) (Hopper and Turton, 2001), BF + metaheuristics (Burke et al., 2009), the hybrid algorithm (Iori et al., 2003), the BLD* algorithm (Lesh et al., 2005), SPGAL (Bortfeld, 2006) and HRP (Huang et al., 2007), SVC, GRASP, and so on. Although results of some algorithms are not reported because of different machine types and running-time limits, from comparisons in Burke et al. (2009), Alvarez-Valdes et al. (2008) and Belov et al. (2008), it is known that BF+SA, GRASP and SVC are the better algorithms. So the three algorithms are selected for comparison with ISA.

Values shown in bold letters in Table 2 denote the best results that BSA, GRASP, SVC and ISA can obtain for each instance. For example, for dataset C, BSA, GRASP and ISA obtain the best result (=20). From Table 2, we can observe that ISA obtains 14 best results whereas BSA, SVC and GRASP obtain 1, 10 and 7 best results, respectively. Moreover, ISA obtains the minimum *Ave.* values of *mean* and *Gap*, 82.84 and 0.76, respectively. In addition, we can

Table 7
Computational results on the non-zero-waste instances BWMV.

Instances				SVC	GRASP		ISA		Gap%		
	n	*W*	LB	Mean	Best	Mean	Best	Mean	SVC	GRASP	ISA
C01	20	10	60.3	61.4	61.4	61.4	61.3	**61.3**	1.8	1.8	1.7
	40	10	121.6	122	121.9	121.9	121.8	**121.8**	0.3	0.2	0.2
	60	10	187.4	188.6	188.6	188.6	188.6	188.6	0.6	0.6	0.6
	80	10	262.2	262.6	262.6	262.6	262.6	262.6	0.2	0.2	0.2
	100	10	304.4	**304.9**	305	305	304.9	**304.9**	0.2	0.2	0.2
C02	20	30	19.7	**19.8**	19.8	**19.8**	19.8	19.9	0.5	0.5	1.0
	40	30	39.1	39.1	39.1	39.1	39.1	39.1	0.0	0.0	0.0
	60	30	60.1	**60.1**	60.3	60.3	60.1	**60.1**	0.0	0.3	0.0
	80	30	83.2	**83.2**	83.3	83.3	83.2	**83.2**	0.0	0.1	0.0
	100	30	100.5	**100.5**	100.6	100.6	100.5	**100.5**	0.0	0.1	0.0
C03	20	40	157.4	164.6	163.5	**163.5**	163.7	164.0	4.6	3.9	4.2
	40	40	328.8	333.9	334.2	334.2	333.4	**333.8**	1.6	1.6	1.5
	60	40	500	506.9	506.6	506.6	505.4	**505.8**	1.4	1.3	1.2
	80	40	701.7	710.1	709.7	709.7	708.8	**709.2**	1.2	1.1	1.1
	100	40	832.7	839.9	840.2	840.2	837.4	**837.8**	0.9	0.9	0.6
C04	20	100	61.4	63.8	63.3	**63.3**	63.6	63.9	3.9	3.1	4.1
	40	100	123.9	126.2	126.2	126.2	125.5	**126.1**	1.9	1.9	1.8
	60	100	193	195.6	196.6	196.6	195.1	**195.5**	1.3	1.9	1.3
	80	100	267.2	270.5	272	272	269.5	**269.8**	1.2	1.8	1.0
	100	100	322	325.3	327.3	327.3	324.1	**324.6**	1.0	1.6	0.8
C05	20	100	512.2	537.9	533.9	**533.9**	534.2	534.6	5.0	4.2	4.4
	40	100	1053.8	1076.4	1074.4	1074.4	1073.3	**1073.6**	2.1	2.0	1.9
	60	100	1614	1647.6	1645.5	1645.5	1642.7	**1643.4**	2.1	2.0	1.8
	80	100	2268.4	**2288.9**	2290.5	2290.5	2288.7	2289.0	0.9	1.0	0.9
	100	100	2617.4	2653.5	2651.1	2651.1	2642.5	**2644.4**	1.4	1.3	1.0
C06	20	10	159.9	169.6	167.2	**167.2**	168.7	169.6	6.1	4.6	6.1
	40	10	323.5	**332.6**	333.4	333.4	332.2	334.0	2.8	3.1	3.2
	60	10	505.1	**517.2**	519.8	519.9	517.8	519.0	2.4	2.9	2.8
	80	10	699.7	**714.7**	718.3	718.4	713.8	715.5	2.1	2.7	2.3
	100	10	843.8	**860.6**	865.1	865.1	859.8	861.1	2.0	2.5	2.1
C07	20	30	490.4	501.9	501.9	501.9	501.9	501.9	2.3	2.3	2.3
	40	30	1049.7	1059.9	1059	**1059**	1059.0	**1059.0**	1.0	0.9	0.9
	60	30	1515.9	1530	1529.6	**1529.6**	1529.6	**1529.6**	0.9	0.9	0.9
	80	30	2206.1	**2222.1**	2222.2	2222.2	2222.1	**2222.1**	0.7	0.7	0.7
	100	30	2627	**2644**	2644	2644	2644.7	2645.4	0.6	0.6	0.7
C08	20	40	434.6	461.2	458.3	**458.3**	457.3	458.6	6.1	5.5	5.5
	40	40	922	956.5	954.3	954.3	950.1	**951.9**	3.7	3.5	3.2
	60	40	1360.9	1403.5	1405	1405	1395.6	**1399.4**	3.1	3.2	2.8
	80	40	1909.3	1965	1971.5	1971.5	1950.1	**1954.7**	2.9	3.3	2.4
	100	40	2362.8	2425	2436.8	2436.8	2406.5	**2410.8**	2.6	3.1	2.0
C09	20	100	1106.8	1106.8	1106.8	1106.8	1106.8	1106.8	0.0	0.0	0.0
	40	100	2189.2	2190.6	2190.6	2190.6	2190.6	2190.6	0.1	0.1	0.1
	60	100	3410.4	3410.4	3410.4	3410.4	3410.4	3410.4	0.0	0.0	0.0
	80	100	4578.6	4588.1	4588.1	4588.1	4588.1	4588.1	0.2	0.2	0.2
	100	100	5430.5	5434.9	5434.9	5434.9	5434.9	5434.9	0.1	0.1	0.1
C10	20	100	337.8	351.5	350.5	350.5	350.2	**350.4**	4.1	3.8	3.7
	40	100	642.8	667	664.4	664.4	663.1	**664.0**	3.8	3.4	3.3
	60	100	911.1	936.6	934.6	934.7	931.3	**933.1**	2.8	2.6	2.4
	80	100	1177.6	1212.4	1209.9	1209.9	1201.6	**1204.1**	3.0	2.7	2.3
	100	100	1476.5	1514	1512.3	1512.3	1501.1	**1504.2**	2.5	2.4	1.9
Ave.				1043.19	1043.33	1043.34	1040.74	**1041.53**	1.80	1.77	**1.66**

observe that GRASP performs well for small instances, while SVC and ISA perform well for large instances. In all, ISA performs better than BSA, GRASP and SVC.

Table 3 shows the results of zero-waste data sets N. For N, some authors have reported computational results of their algorithms, such as BF+metaheuristic and GRASP. From comparisons in Burke et al. (2009) and Alvarez-Valdes et al. (2008), we know that BF+SA and GRASP are better algorithms. At the same time, we also report results of SVC for N.

From Table 3, we can observe that compared to BSA (0), SVC (3), and GRASP (0), ISA produces 9 best results. Besides, ISA obtains the minimum *Ave.* values of *mean* and *Gap*, 178.18 and 0.41, respectively. In addition, we can observe that ISA performs well for all instances.

Table 4 shows the results of zero-waste data sets NT. For NT, many researchers have reported computational results of their algorithms; BLD*, SVC, GRASP and others. Although we do not report the results of BLD* because of different machine types and running-time limits, we know that BF + SA, GRASP and SVC are the better algorithms, from comparisons in Alvarez-Valdes et al. (2008) and Belov et al. (2008), so we compare them with ISA only.

From Table 4, SVC and GRASP obtain 25 and 19 best results, respectively, whereas ISA obtains 30 best results. Moreover, ISA obtains the lowest *Ave.* values of *mean* and *Gap*. In addition, we can observe that GRASP performs well for small instances, while SVC and ISA perform well for large instances. In particular, ISA performs better for the largest instances. Therefore, ISA outperforms BSA, GRASP and SVC, on average.

Table 8
Computational results of Nice and Path.

Instance				SVC	GRASP		ISA		Gap%		
	n	*W*	LB	Mean	Best	Mean	Best	Mean	SVC	GRASP	ISA
Nice1	25	1000	1000	1037	1034	**1034**	1035	1040.7	3.7	3.4	4.1
Nice2	50	1000	1001	**1038**	1047	1047	1043	1047.2	3.7	4.6	4.6
Nice3	100	1000	1001	**1035**	1041	1041	1029	1036.5	3.4	4.0	3.5
Nice4	200	1000	1001	**1026**	1037	1037	1030	1030.9	2.5	3.6	2.9
Nice5	500	1000	1000	1017	1024	1024	1015	**1015.0**	1.7	2.4	1.5
Nice6	1000	1000	999	1014	1020	1020	1011	**1011.0**	1.5	2.1	1.2
Nice1t	1000	1000	1001	1015	1026	1026	1011	**1011.0**	1.4	2.5	1.0
	1000	1000	1001	1016	1022	1022	1010	**1010.0**	1.5	2.1	0.9
	1000	1000	1000	1013	1020	1020	1011	**1011.0**	1.3	2.0	1.1
	1000	1000	1000	1013	1019	1019	1010	**1010.0**	1.3	1.9	1.0
	1000	1000	1000	1014	1022	1022	1010	**1010.0**	1.4	2.2	1.0
	1000	1000	1001	1014	1020	1020	1010	**1010.0**	1.3	1.9	0.9
	1000	1000	1000	1014	1022	1022	1010	**1010.0**	1.4	2.2	1.0
	1000	1000	1001	1016	1021	1021	1012	**1012.0**	1.5	2.0	1.1
	1000	1000	1000	1017	1022	1022	1012	**1012.0**	1.7	2.2	1.2
	1000	1000	1001	1016	1027	1027	1012	**1012.0**	1.5	2.6	1.1
Nice2t	2000	1000	1001	1008	1016	1016	1006	**1006.0**	0.7	1.5	0.5
	2000	1000	1001	1011	1015	1015	1005	**1005.0**	1.0	1.4	0.4
	2000	1000	1000	1008	1016	1016	1007	**1007.0**	0.8	1.6	0.7
	2000	1000	1000	1007	1014	1014	1006	**1006.0**	0.7	1.4	0.6
	2000	1000	1000	1008	1015	1015	1006	**1006.0**	0.8	1.5	0.6
	2000	1000	1000	**1002**	1016	1016	1005	1005.0	0.2	1.6	0.5
	2000	1000	1001	**1007**	1016	1016	1007	**1007.0**	0.6	1.5	0.6
	2000	1000	1001	**1006**	1014	1014	1006	**1006.0**	0.5	1.3	0.5
	2000	1000	1001	1008	1016	1016	1007	**1007.0**	0.7	1.5	0.6
	2000	1000	1001	1009	1016	1016	1007	**1007.0**	0.8	1.5	0.6
Nice5t	5000	1000	1000	**1003**	1010	1010	1003	**1003.0**	0.3	1.0	0.3
	5000	1000	1001	1005	1011	1011	1003	**1003.0**	0.4	1.0	0.2
	5000	1000	1001	**1002**	1010	1010	1003	1003.0	0.1	0.9	0.2
	5000	1000	1000	1005	1009	1009	1002	**1002.0**	0.5	0.9	0.2
	5000	1000	1001	1006	1011	1011	1003	**1003.0**	0.5	1.0	0.2
	5000	1000	1000	**1001**	1009	1009	1002	1002.0	0.1	0.9	0.2
	5000	1000	1001	1004	1011	1011	1003	**1003.0**	0.3	1.0	0.2
	5000	1000	1000	1004	1011	1011	1002	**1002.0**	0.4	1.1	0.2
	5000	1000	1001	1004	1010	1010	1003	**1003.0**	0.3	0.9	0.2
	5000	1000	1000	1005	1010	1010	1003	**1003.0**	0.5	1.0	0.3
Path1	25	1000	1001	1042	1042	1042	1042	1042.0	4.1	4.1	4.1
Path2	50	1000	1000	**1014**	1019	1019	1012	1014.7	1.4	1.9	1.5
Path3	100	1000	1000	**1022**	1027	1027	1020	1022.6	2.2	2.7	2.3
Path4	200	1000	1002	1018	1023	1023	1017	**1017.7**	1.6	2.1	1.6
Path5	500	1000	1000	1022	1034	1034	1020	**1020.0**	2.2	3.4	2.0
Path6	1000	1000	1002	1018	1026	1026	1011	**1011.0**	1.6	2.4	0.9
Path1t	1000	1000	999	1011	1019	1019	1007	**1007.0**	1.2	2.0	0.8
	1000	1000	1001	1010	1018	1018	1006	**1006.0**	0.9	1.7	0.5
	1000	1000	1001	1013	1018	1018	1008	**1008.0**	1.2	1.7	0.7
	1000	1000	1000	1009	1016	1016	1006	**1006.0**	0.9	1.6	0.6
	1000	1000	1003	1017	1024	1024	1010	**1010.0**	1.4	2.1	0.7
	1000	1000	1002	1013	1018	1018	1010	**1010.0**	1.1	1.6	0.8
	1000	1000	999	1012	1019	1019	1008	**1008.0**	1.3	2.0	0.9
	1000	1000	1000	1012	1020	1020	1008	**1008.0**	1.2	2.0	0.8
	1000	1000	999	1012	1019	1019	1006	**1006.0**	1.3	2.0	0.7
	1000	1000	1002	1011	1018	1018	1008	**1008.0**	0.9	1.6	0.6
Path2t	2000	1000	1000	1009	1015	1015	1006	**1006.0**	0.9	1.5	0.6
	2000	1000	1002	1010	1016	1016	1007	**1007.0**	0.8	1.4	0.5
	2000	1000	1000	1011	1015	1015	1006	**1006.0**	1.1	1.5	0.6
	2000	1000	999	1007	1014	1014	1003	**1003.0**	0.8	1.5	0.4
	2000	1000	1002	1012	1018	1018	1008	**1008.0**	1.0	1.6	0.6
	2000	1000	1002	1011	1016	1016	1007	**1007.0**	0.9	1.4	0.5
	2000	1000	998	1007	1011	1011	1004	**1004.0**	0.9	1.3	0.6
	2000	1000	998	1010	1014	1014	1004	**1004.0**	1.2	1.6	0.6
	2000	1000	1001	1010	1017	1017	1008	**1008.0**	0.9	1.6	0.7
	2000	1000	1003	**1009**	1018	1018	1009	**1009.0**	0.6	1.5	0.6
Path5t	5000	1000	1000	**1002**	1010	1010	1003	1003.0	0.2	1.0	0.3
	5000	1000	998	1003	1009	1009	1001	**1001.0**	0.5	1.1	0.3
	5000	1000	1000	**1002**	1011	1011	1003	1003.0	0.2	1.1	0.3
	5000	1000	995	998	1006	1006	997	**997.0**	0.3	1.1	0.2
	5000	1000	1004	**1005**	1016	1016	1006	1006.0	0.1	1.2	0.2
	5000	1000	1000	1003	1009	1009	1002	**1002.0**	0.3	0.9	0.2
	5000	1000	998	**1001**	1009	1009	1001	**1001.0**	0.3	1.1	0.3
	5000	1000	996	**998**	1007	1007	999	999.0	0.2	1.1	0.3

Table 8 (*continued*)

Instance				SVC	GRASP		ISA		Gap%		
	n	*W*	LB	Mean	Best	Mean	Best	Mean	SVC	GRASP	ISA
	5000	1000	997	**999**	1007	1007	999	**999.0**	0.2	1.0	0.2
	5000	1000	1002	**1004**	1013	1013	1004	**1004.0**	0.2	1.1	0.2
Ave.				1009.75	1016.63	1016.63	1007.19	**1007.36**	0.96	1.65	**0.72**

Table 9
Computational results the non-zero-waste instances ZDF.

Instance				SVC	GRASP		ISA		Gap%		
	n	*W*	LB	Mean	Best	Mean	Best	Mean	SVC	GRASP	ISA
zdf1	580	100	330	331	333	333	330	**330.0**	0.3	0.9	0.0
zdf2	660	100	357	358	360	360	357	**357.0**	0.3	0.8	0.0
zdf3	740	100	384	385	387	387	384	**384.0**	0.3	0.8	0.0
zdf4	820	100	407	408	410	410	407	**407.0**	0.2	0.7	0.0
zdf5	900	100	434	**434**	437	437	434	**434.0**	0.0	0.7	0.0
zdf6	1532	3000	4872	5085	5251	5251	5049	**5081.8**	4.4	7.8	4.3
zdf7	2432	3000	4852	**5083**	5163	5163	5083	5084.7	4.8	6.4	4.8
zdf8	2532	3000	5172	**5386**	5544	5544	5549	5549.0	4.1	7.2	7.3
zdf9	5032	3000	5172	5468	5476	5476	5404	**5404.0**	5.7	5.9	4.5
zdf10	5064	6000	5172	5462	5570	5570	5419	**5419.0**	5.6	7.7	4.8
zdf11	7564	6000	5172	5516	5562	5562	5419	**5419.0**	6.7	7.5	4.8
zdf12	10,064	6000	5172	5651	–	–	5454	**5454.0**	9.3	–	5.5
zdf13	15,096	9000	5172	5600	–	–	5415	**5415.0**	8.3	–	4.7
zdf14	25,032	3000	5172	5468	–	–	5286	**5286.0**	5.7	–	2.2
zdf15	50,032	3000	5172	5960	–	–	5172	**5172.0**	15.2	–	0.0
zdf16	75,032	3000	5172	5931	–	–	5172	**5172.0**	14.7	–	0.0
Ave.				3907.88	–	–	3770.88	**3773.03**	5.34	–	**2.67**

Table 10
Statistical results on all the instances.

		C	N	Babu	*N* and *T*	Nice and path	CX	ZDF	2sp	BW	Total
Instances		21	13	1	70	72	7	16	38	500	738
ON	GRASP	8	2	1	9	0	3	0	21	48	92
	SVC	7	3	1	6	0	3	1	21	52	94
	ISA	8	5	1	8	0	4	7	19	54	**106**

Table 5 shows the results on zero-waste data sets CX, which include an extra large instance ($n = 15000$). For CX, some authors have reported computational results of their algorithms such as BLF (Pinto and Oliveira, 2005) and GRASP. We have also calculated results of SVC, GRASP and ISA for CX. From Table 5, ISA performs worst for the smallest instance 50cx and also it cannot obtain the lowest *Ave.* values of *mean* and *Gap*, but it obtains three best results, compared to SVC (1) and GRASP (0). Moreover, ISA obtains the optimal solution for 1000cx, which means it performs better for large instances.

5.2. Computational results on non-zero-waste problem instances

In many practical applications, the optimal solution often includes some wasted regions. So non-zero-waste problem instances are more general. Table 6 shows the results of non-zero-waste data sets 2sp. One characteristic of 2sp is that the size of problem instance (n) is very small, less than 200. For the 2sp, many authors have reported computational results of their algorithms: the hybrid algorithm (Iori et al., 2003), BLD$*$, GRASP, and so on. Although results of some algorithms are not reported in this paper because of different machine types and running-time limits, from comparisons with Belov et al. (2008), we know that GRASP and SVC are better algorithms, and so they are selected for comparison with ISA.

Computational results of SVC, GRASP and ISA for 2sp are reported in Table 6. We can observe that the number of best results obtained by SVC is 4, whereas GRASP obtains seven best results and the number for ISA is 4. GRASP obtains the lowest *Ave.* value of *Gap*, 2.31. Therefore, ISA performs worse than GRASP for small instances. However, ISA obtains the lowest *Ave.* value of *mean* (1538.64), so ISA is still efficient for data set 2sp.

Table 7 shows the results of non-zero-waste data sets BWMV. One characteristic of BWMV is that the size of problem instance (n) is up to 100. Since we know from comparisons in Alvarez-Valdes et al. (2008) and Belov et al. (2008) that GRASP and SVC are better algorithms, they are selected for comparison with ISA.

The average computational results of SVC, GRASP and ISA for each BWMV instance of the same size are reported in Table 7. From Table 7, we can observe that SVC produces 12 best results, while the number of best results obtained by GRASP is 8, and the number of best results obtained by ISA is 29. Besides, ISA obtains the lowest *Ave.* values of *mean* and *Gap*, 1041.53 and 1.66, respectively. Therefore, ISA performs better than GRASP and SVC. In addition, we can observe that GRASP performs well for small instances, while SVC and ISA perform well for large instances.

Table 8 shows the results of non-zero-waste data sets Nice and Path. Nice and Path are floating-point data sets. Nice and Path are regarded as non-zero-waste instances because the integer data are obtained by multiplying the original data by 10 and rounding to the nearest integer (Alvarez-Valdes et al., 2008). The characteristics of Nice and Path are that the size of problem instance (n) is from 25 to 5000, and include data of different types. For Nice1–Nice6 and Path1–Path6, some researchers have reported their results, examples being BF + metaheuristic and GRASP. Some researchers have used Nice1t–Nice5t and Path1t–Path5t to compare performances of different algorithms.

Computational results of SVC, GRASP and ISA for Nice and Path are reported in Table 8. From Table 8, we can observe that ISA produces 58 best results, compared to only 19 of SVC and 1 of GRASP. Besides, ISA obtains the lowest *Ave.* values of *mean* and *Gap*, 1007.36 and 0.72, respectively. Therefore, ISA performs better than GRASP and SVC. In addition, we can observe that GRASP only performs well for one small instance, while SVC and ISA perform well for large instances.

Table 9 shows results of non-zero-waste large data sets ZDF, generated by combining zero-waste and non-zero-waste data. Moreover, their n value is very large; in particular, the size of zdf16 is 75,032, which is the largest instance so far. Generally, it is difficult to obtain the optimal solution of an instance as large as ZDF within a reasonable time.

ZDF data set is tested by SVC, GRASP and ISA. Computational results are reported in Table 9. The symbol "—" denotes that GRASP cannot return the right results by the GRASP executable programme. From Table 9, we can observe that the number of best results obtained by SVC is 3, GRASP produces no best result, and the number for ISA is 14. ISA obtains the lowest *Ave.* values of *mean* and *Gap*, 3773.03 and 2.67, respectively. Therefore, ISA is better than GRASP and SVC for large instances.

According to the computational results in Tables 2–9, we report the statistics of results of GRASP, SVC and ISA in Table 10. *ON* denotes the number of optimal solutions obtained by an algorithm. From Table 10, we can observe that three algorithms obtain optimal solutions for data set Babu. For small data sets NT and 2sp, GRASP can obtain a higher *ON* than SVC and ISA. SVC can obtain a higher *ON* than ISA for 2sp. However, ISA can obtain a higher *ON* than GRASP and SVC for most data sets.

6. Conclusions

A two-stage intelligent search algorithm for the orthogonal strip packing problem is presented in this paper. This algorithm includes three new ideas: a novel scoring rule, a data structure that determines the available spaces, and a Multi-start strategy. Local search is used to find a good solution and a simulated annealing algorithm is used to enhance the search capability of the algorithm and further improve the solutions. Computational results have shown that ISA outperforms algorithms currently viewed as excellent. It performs better, particularly for large problem instances. From the experiment of parameter selection, ISA is more stable and efficient. So the packing software based on ISA may be of great practical value for achieving rational loading layouts of rectangular objects in engineering fields. Future work is to further improve the performance of this algorithm and extend it to solve three-dimensional packing problems.

Acknowledgment

The authors would like to thank Prof. Ramon Alvarez-Valdes, Dr. G. Belov and Dr. C.L. Mumford for their executable programs and test data sets. The authors would like to express our appreciation to the reviewers in making valuable comments and suggestions to the paper. Their comments have improved and enriched the quality of the paper immensely. This work was supported by the National Nature Science Foundation of China (Grant No. 60773126).

References

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

Baker, B.S., Coffman Jr., E.G., Rivest, R.L., 1980. Orthogonal packings in two dimensions. SIAM Journal on Computing 9 (4), 846–855.

Beasley, J.E., 1985a. An exact two-dimensional non-guillotine cutting tree search procedure. Operations Research 33, 49–64.

Beasley, J.E., 1985b. Algorithms for unconstrained two-dimensional guillotine cutting. Journal of the Operational Research Society 36, 297–306.

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

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

Bortfeldt, A., 2006. A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces. European Journal of Operational Research 172 (3), 814–837.

Burke, E.K., Kendall, G., Whitwell, G., 2004. A new placement heuristic for the orthogonal stock-cutting problem. Operations Research 52 (4), 655–671.

Burke, E.K., Kendall, G., Whitwell, G., 2009. A simulated annealing enhancement of the best-fit heuristic for the orthogonal stock cutting problem. INFORMS Journal on Computing 21 (3), 505–516.

Chazelle, B., 1983. The bottom-left bin packing heuristic: an efficient implementation. IEEE Transaction on Computers 32 (8), 697–707.

Christofides, N., Hadjiconstantinou, E., 1995. An exact algorithm for orthogonal 2-D cutting problems using guillotine cuts. European Journal of Operational Research 83, 21–38.

Christofides, N., Whitlock, C., 1997. An algorithm for two-dimensional cutting problems. Operations Research 25, 30–44.

Cui, Y., Yang, Y., Cheng, X., Song, P., 2008. A recursive branch-and- bound algorithm for the rectangular guillotine strip packing problem. Computers and Operations Research 35 (4), 1281–1291.

Dagli, C.H, Hajakbari, A., 1990. Simulated annealing approach for solving stock cutting problem. In: Proceedings of IEEE International Conference on Systems, Man, Cybernetics, Los Angeles. IEEE, Washington, DC, pp. 221–223.

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

Dowsland, K.A., Dowsland, W.B., 1992. Packing problems. European Journal of Operational Research 56, 2–14.

Dowsland, K.A., Herbert, E.A., Kendall, G., Burke, E.K., 2006. Using tree search bounds to enhance a genetic algorithm approach to two rectangle packing problems. European Journal of Operational Research 168, 390–402.

Fekete, S.P., Schepers, J., van der Veen, J.C., 2007. An exact algorithm for higher-dimensional orthogonal packing. Operations Research 55, 569–590.

Gonçalves, J.F., 2007. A hybrid genetic algorithm-heuristic for a two-dimensional orthogonal packing problem. European Journal of Operational Research 183, 1212–1229.

Hopper, E., Turton, B.C.H., 2001. An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. European Journal of Operational Research 128, 34–57.

Hifi, M., M'Hallah, R., 2005. An exact algorithm for constrained two-dimensional two-staged cutting problems. Operations Research 53 (1), 140–150.

Hifi, M., M'Hallah, R., 2003. A hybrid algorithm for the two-dimensional layout problem: the cases of regular and irregular shapes. International Transactions in Operational Research 10, 1–22.

Hochbaum, D.S., Maass, W., 1985. Approximation schemes for covering and packing problems in image processing and VLSI. Journal of ACM 32 (1), 130–136.

Huang, W., Chen, D., Xu, R., 2007. A new heuristic algorithm for rectangle packing. Computers and Operations Research 34 (11), 3270–3280.

Iori, M., Martello, S., Monaci, M., 2003. Metaheuristic algorithms for the strip packing problem. In: Paradolos, P.M., Korotkith, V. (Eds.), Optimization and Industry: New Frontiers. The Netherlands, Kluwer Academic Publishers, pp. 159–179.

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

Kenmochi, M., Takashi, I., Koji, N., Mutsunori, Y., Hiroshi, N., 2009. Exact algorithms for the two-dimensional strip packing problem with and without rotations. European Journal of Operational Research 198, 73–83.

Lai, K.K., Chan, J.W.M., 1996. Developing a simulated annealing algorithm for the cutting stock problem. Computers and Industrial Engineering 32 (1), 115–127.

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

Leung, S., Zhang, D., 2011. A fast layer-based heuristic for non-guillotine strip packing. Expert Systems with Applications 38 (10), 13032–13042.

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.

Lodi, A., Martello, S., Monaci, M., 2002. Two-dimensional packing problems: a survey. European Journal of Operational Research 141, 241–252.

Martello, S., Monaci, M., Vigo, D., 2003. An exact approach to the strip packing problem. INFORMS Journal on Computing 15 (3), 310–319.

Ortmann, F.G., Ntene, N., van Vuuren, J.H., 2010. New and improved level heuristics for the rectangular strip packing and variable-sized bin packing problems. European Journal of Operational Research 203 (2), 306–315.

Pinto, E., Oliveira, J.F., 2005. Algorithm based on graphs for the non-guillotinable two-dimensional packing problem. in: Second ESICUP Meeting, Southampton.

Ramesh Babu, A., Ramesh Babu, N., 1999. Effective nesting of rectangular parts in multiple rectangular sheets using genetic and heuristic algorithms. International Journal of Production Research 37 (7), 1625–1643.

Valenzuela, C.L., Wang, P.Y., 2001. Heuristics for large strip packing problems with guillotine patterns: an empirical study. In: Proceedings of the 4th Metaheuristics International Conference. University of Porto, Porto, Portugal, pp. 417–421.

Wäscher, G., Haußner, H., Schumann, H., 2007. An improved typology of cutting and packing problems. European Journal of Operational Research 183, 1109–1130.

Wei, L., Zhang, D., Chen, Q., 2009. A least wasted first heuristic algorithm for the rectangular packing problem. Computers and Operations Research 36 (5), 1608–1614.

Zhang, D., Kang, Y., Deng, A., 2006. A new heuristic recursive algorithm for the strip rectangular packing problem. Computers and Operations Research 33 (8), 2209–2217.

Zhang, D., Han, S., Ye, W., 2008. A bricklaying heuristic algorithm for the orthogonal rectangular packing problem. Chinese Journal of Computers 23 (3), 509–515.