European Journal of Operational Research 240 (2015) 43–53

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

Sequential heuristic for the two-dimensional bin-packing problem

Yi-Ping Cui, Yaodong Cui *, Tianbing Tang

College of Computer and Electronic Information, Guangxi University, Nanning 530004, China

ARTICLE INFO

Article history:
Received 5 February 2013
Accepted 20 June 2014
Available online 1 July 2014

Keywords:
Packing
2D bin-packing
Sequential value correction

ABSTRACT

A heuristic approach for the two-dimensional bin-packing problem is proposed. The algorithm is based on the sequential heuristic procedure that generates each pattern to produce some items and repeats until all items are produced. Both guillotine and non-guillotine patterns can be used. Each pattern is obtained from calling a pattern-generation procedure, where the objective is to maximize the pattern value. The item values are adjusted after the generation of each pattern using a value correction formula. The algorithm is compared with five published algorithms, using 50 groups of benchmark instances. The results indicate that the algorithm is the most efficient in improving solution quality.

1. Introduction

In the industries such as wood, metal and glass, it is often necessary to produce a set of rectangular items $i \in I = \{1, \ldots, m\}$ with length l_i and height h_i (the demand of each item type is 1), from larger bins with length L and height H. This can be modeled as the two-dimensional bin-packing problem (2BP). The objective is to minimize material input (number of bins). It is assumed that each item must be packed parallel to the edge of a bin, in either the given orientation or 90 degrees rotated. A choice should be made between either using guillotine patterns or free (non-guillotine) ones.

The 2BP can be classified according to the following two rules (Lodi, Martello, & Daniele, 1999):

(1) Orientation: the items may have either a fixed orientation or 90 degrees rotated.
(2) Type of cuts: whether or not the cutting patterns should meet the requirement of guillotine cuts.

Considering the combination of these two rules, the 2BP can be classified into four types:

(1) 2BPOG: Items are oriented (O) and guillotine cuts are required (G).
(2) 2BPRG: Items may be rotated by 90 degrees (R) and guillotine cuts are required (G).
(3) 2BPOF: Items are oriented (O) and cutting is free (F).
(4) 2BPRF: Items may be rotated by 90 degrees (R) and cutting is free (F).

The solution of the 2BP is a cutting plan that contains a set of cutting patterns, each of which has specified items layout. The (usage) frequency of each pattern is one, or equivalently, the number of patterns is equal to the number of bins used.

In this paper, a sequential value correction heuristic (SVC2BPR) for 2BPR (both 2BPRG and 2BPRF) is presented. The algorithm is denoted as SVC2BPRG when guillotine patterns are used and as SVC2BPRF when free patterns are allowed. It generates a specified number of cutting plans, using different item values. The cutting plan that yields the minimum number of bins is selected as the solution. Each pattern in the current cutting plan is obtained from calling a pattern-generation procedure, where the objective is to maximize the pattern value, that is, the total value of the included items. Initially the item values are equal to their areas. To diversify the cutting plans, the values of the items included in the current pattern are corrected after each pattern being generated based on the size of the items and the material utilization of the pattern, using a formula similar to those in the literature (Belov & Scheithauer, 2007; Belov, Scheithauer, & Mukhacheva, 2008; Cui, Yang, & Chen, 2013). 500 benchmark instances in 50 groups are used to test the performance of the algorithm. The computational results show that the algorithm can achieve better solutions in a reasonable amount of time, compared to five algorithms published in the literature (Boschetti & Mingozzi, 2003; Charalambous & Fleszer, 2011; Harwig, Barnes, & James, 2006; Hayek, Moukrim, & Negre, 2008; Polyakovsky & M'Hallah, 2009).

The paper makes contributions possibly from the following aspects:

* Corresponding author. Tel.: +86 07713232214.
E-mail addresses: yipingcuiv@gmail.com (Y.-P. Cui), ydcui@263.net (Y. Cui).

http://dx.doi.org/10.1016/j.ejor.2014.06.032
0377-2217/

(1) It may be the first to solve the 2BPR with a sequential procedure that is based on value correction of the items
(2) It may be the first to use the patterns with the specified geometric structure (described later in Section 3.1) to solve the 2BPRG. This type of patterns is easy to use in practice.
(3) It proposes a look-ahead strategy adequate for improving the quality of the free patterns used in this paper.
(4) The proposed algorithm is able to improve the average solution qualities of both 2BPRG and 2BPRF benchmark instances.

The next sections are arranged as follows. Literature is reviewed in Section 2. The SVC2BPR is presented in Section 3. In Section 4, benchmark instances are used to evaluate the algorithm's effectiveness in material input minimization, and the computational results are compared with those of five published algorithms. Conclusions are given in the last section.

2. Literature review

A thorough introduction to the 2BP can be seen in Lodi, Martello, and Vigo (2002A). Recent approaches on the 2BP can also be found in Wei, Oon, and Zhu (2013), Liao and Hsu (2013) and Han, Bennell, and Zhao (2013).

In Lodi et al. (1999), a new heuristic algorithm is introduced for each one of the four 2BP types. Also, a unified tabu search approach is adapted to a specific problem by changing the heuristic used to explore the neighborhood. The heuristic algorithm and the search are evaluated in the paper.

In Dell'Amico, Martello, and Vigo (2002) and Clautiaux, Jouglet, and Hayek (2007), lower bound for the 2BP is discussed. Lower bounds are useful in evaluating the efficiency of the algorithms. They can also be used to reduce the computation time because the searching process can be terminated once the solution value reaches the lower bound.

Algorithms for the 2BPO (2BPOG or 2BPOF) may be extended to deal with the 2BPR, but the extending may be not straightforward. Considering that the algorithm proposed in this paper deals with the 2BPR only. The review in this section is focused on those papers that involve the 2BPR.

From the comment on exact 2BP algorithms in the literature (Polyakovsky & M'Hallah, 2009), the following conclusions can be drawn. The 2BPR is NP-hard. Solving it exactly is generally impossible; especially when the instance contains a large number of items. The literature review in this paper will concentrate on heuristic algorithms. The reader is referred to Lodi, Martello, and Vigo (2002B) for more detailed descriptions of the exact methods.

2.1. Approaches for 2BPRG

Three constructive heuristics: first-fit insertion, best-fit insertion and critical-fit insertion are proposed in Fleszar (2013). The heuristics use tree structures to represent guillotine patterns and process by insertion one item at a time in a partial solution. Central to all heuristics are a new procedure for enumerating possible insertions and a new fitness criterion for choosing the best insertion. All new heuristics have quadratic worst-case computational complexity except for the critical-fit insertion heuristic that has a cubic complexity. The efficiency and effectiveness of the proposed heuristics are demonstrated by comparing their empirical performance on a standard benchmark data set against other published approaches.

Stochastic neighborhood structures (SNS) are combined with heuristic algorithm to solve two-stage and three-stage 2BP with guillotine cuts in Chan, Alvelos, Silva, and Carvalho (2011). The SNS comprises three random neighborhood structures based on modifying the current sequence of items. According to the computation results, the SNS provides solutions within a small percentage of the optimal values, and generally makes significant improvements in cutting stock instances and slight to moderate improvements in bin-packing instances.

The heuristic in Charalambous and Fleszer (2011) constructs a solution by packing a bin at a time. Central to the adopted solution scheme is the principle of average-area sufficiency proposed by the authors for guiding the selection of items to fill a bin. The algorithm is tested on a set of standard benchmark instances and compared with existing heuristics producing the best-known results. The results presented attest to the efficiency of the proposed scheme.

Polyakovsky and M'Hallah (2009) proposed a new guillotine bottom left (GBL) constructive heuristic and its agent-based (A-B) implementation to solve the 2BP with guillotine cuts. The A-B system consists of active agents dynamically interacting in real time to jointly fill the bins while each agent is driven by its own parameters, decision process and fitness assessment. The GBL is the basis of each agent. It places items in the bottom-left most available position of the bin and decides the direction of the first cut of the strip containing the packed item. It subsequently updates the two unoccupied areas of the bin and continues the filling process until no unpacked item fits in the bin. A new bin is created every time the current one is saturated. This process is repeated until all items are packed.

A straightforward heuristic is proposed in Mack and Bortfelt (2012). It is based on a method for the container loading problem following a wall building approach and on a method for the one-dimensional bin-packing problem.

The algorithm SVC2BPRG in this paper is based on the sequential value correction procedure that generates a specified number of cutting plans, from which the best one is selected as the solution. The patterns in a cutting plan are generated sequentially. This makes it convenient to consider practical restrictions and objectives when necessary. The guillotine patterns used may be useful to simplify the cutting/packing process because of their simple geometric structure. Compared to two published 2BPRG algorithms that produced the best known results, the SVC2BPRG proved to be more efficient (in improving solution quality) in most of the test instances.

2.2. Approaches for 2BPRF

In Bengtsson (1982), two 2BPRF problems are considered. One is finding a way to pack an arbitrary collection of rectangular pieces into an open-ended, rectangular bin so as to minimize the height to which the pieces fill the bin (referred to as the strip packing problem in the literature). The other is to allocate the rectangular pieces into given rectangular sheets, i.e. bins of fixed width and height. Approximation algorithms are given where the solutions are achieved with heuristic search methods.

Boschetti and Mingozzi (2003) described new lower and upper bounds for a general version of the 2BP problem, in which case, each item is associated with an input parameter specifying if it has a fixed orientation or it can be rotated by 90 degrees.

Harwig et al. (2006) implemented an adaptive tabu search procedure that controls the partitioning, ordering and orientation features of a two-dimensional orthogonal packing solution. They detailed an effective fine-grained objective function for cutting and packing problems, and presented effective move neighborhoods to find good answers in a short period of time.

In Hayek et al. (2008), the concept of dependent orientation items that have special characteristics is introduced, and the formulation that characterizes these items is given. Then three

pretreatments for the non-oriented version of the problem are proposed. These pretreatments allow finding optimal packing of some items subsets of the given instance. They enable increasing the total area of the items and consequently the continuous lower bound. Finally, a new heuristic method based on a best-fit algorithm adapted to the 2BP problem is presented. Numerical experiments show that the method is competitive with the heuristic and meta-heuristic algorithms proposed in the literature in respect of both the quality of the solution and the computation time.

The proposed algorithm SVC2BPRF differs from the SVC2BPRG in the pattern-generation procedure that uses a beam search method and a greedy rule. Compared to three published 2BPRF algorithms which produced the best known results, the SVC2BPRF proved to be more efficient (in improving solution quality) in most of the test instances.

3. Approach

The following symbols are used in this section:

N	Number of patterns/bins (solution value)
G	Number of generated cutting plans
$G_{\max}$	Max number of generated cutting plans
v_i	Value of item i,$i = 1, \ldots, m$
b_i	Remaining demand of item i, $b_i \in \{0,1\}$, $i = 1, \ldots, m$
y_i	Number of item i in the current pattern, $y_i \in \{0,1\}$, $i = 1, \ldots, m$

Algorithm SVC2BPR follows the following procedure, in which the function GetPattern () is to be described for guillotine patterns in Section 3.1 and for free patterns in Section 3.2; function ValueCorrection () is to be described in Section 3.3.

Step 1: Let $G = 0$, $v_i = l_i h_i$, $b_i = 1$, $i = 1, \ldots, m$.
Step 2: Let $G = G + 1$. If $G > G_{max}$ or $N = \lceil \sum_{i=1}^{m} l_i h_i / (LH) \rceil$ then go to step 9; otherwise initialize the current cutting plan.
Step 3: Call GetPattern () to obtain the current pattern.
Step 4: Add the pattern to the current cutting plan. Set $b_i = b_i - y_i$, $i = 1, \ldots, m$.
Step 5: Call ValueCorrection () to correct the values of the items included in the current pattern.
Step 6: Go to step 3 if $\exists b_i > 0$, $i = 1, \ldots, m$.
Step 7: If the solution value is smaller than that of the best solution obtained so far, save the current plan as the best one.
Step 8: Go to step 2.
Step 9: Output the best solution.

SVC2BPR generates G_{max} cutting plans, from which the best one is selected as the solution. *Steps 3–6* show that the patterns in the current cutting plan are generated sequentially. Each pattern is used to produce some of the items. The current cutting plan is finished when all items are produced. It replaces the best solution obtained so far if an improvement is obtained (*step 7*). In *step 5*, the values of the items are corrected after the generation of each pattern so as to diversify the cutting plans.

3.1. Pattern-generation function for guillotine patterns

The pattern-generation function is called in *step 3* of SVC2BPR to generate each next pattern. Its effectiveness and efficiency are vital to those of the algorithm.

To generate the cutting patterns, a variety of item combinations have to be considered. However, considering all the possible combinations is not tolerable because of the computation time limit. To reduce the computation time and simplify the cutting/packing process, the patterns of specified geometric structure can be considered. We recursively generate patterns where each guillotine cut, horizontal or vertical, separates a strip having the width of one certain item from the remaining stock piece. In next subsections, the geometric structure of the patterns is introduced first, then the generation of strips is presented, and the pattern generation procedure is described at last.

3.1.1. Geometric structure of the patterns

The proposed patterns are based on strips. As shown in Fig. 1, a strip can be either horizontal or vertical. A horizontal strip contains a row of items. The strip width is equal to the maximum of the vertical edges of the included items. A vertical strip contains a column of items. The strip width is the same as the maximum of the horizontal edges of the included items.

General guillotine patterns (G, Seong, & Kang, 2003) are generated without considering geometric constraints other than the guillotine cuts. Although they are useful to improve solution quality, the packing process may be complex and the generation time is often long. To simplify the packing process and reduce computation time, general-block patterns are used in this paper. They have the feature that each cut on the bin produces just one strip except the last one that yields two strips. The strip structure is useful to simplify the packing process, where items in the same strip can be placed one by one along the bottom edge of a horizontal strip

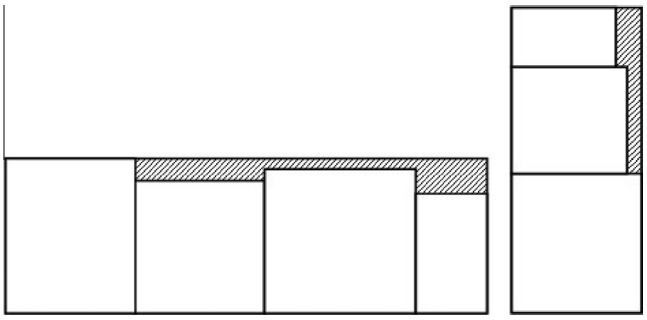

Fig. 1. Strips.

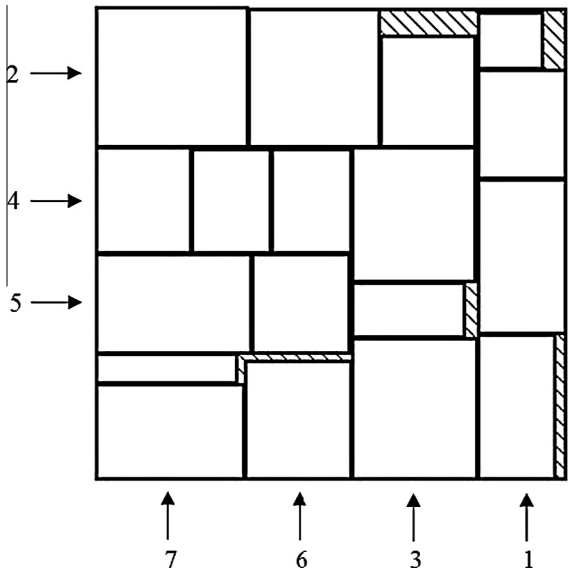

Fig. 2. General block pattern.

or the left edge of a vertical strip. A general block pattern is shown in Fig. 2, where the numbers at the ends of the arrows denote the cutting order of the strips. The pattern contains three horizontal (strips 2, 4 and 5) and four vertical (strips 1, 3, 6 and 7) strips.

It should be noted that general blocks have been used as elements to compose complex cutting patterns in Cui and Zhang (2007), where the number of cuts necessary to produce a strip is often larger than one. The procedure in Cui and Zhang (2007) for determining general blocks cannot be used to generate the patterns for the 2BPRG, because it does not consider the constraint on the number of copies of an item.

3.1.2. Function GetStrip(L)

The strips must be determined before the generation of each pattern. One key characteristic of a strip is its width. The width of a strip can be equal to the length or height of an item. Suppose that there are m items available. Then $2m$ strip types can be considered. Let W_j be the width of strip type j, then $W_j = h_j$ when $j \in \{1,\ldots,m\}$ and $W_j = l_{j-m}$ when $j \in \{m+1,\ldots,2m\}$.

The function GetStrip (L) generates all the strip types with length L. For a particular strip type with width W and length x, let $z(i,x)$ be the maximum value of the strip obtained from considering the first i items, where $i = 1,\ldots,m$, $x = 0,\ldots,L$, and the value of a strip denotes the total value of the included items. Initially let $z(0,x) = 0$ for $x = 0,\ldots,L$, then the $z(i,x)$ values can be obtained from the following standard recursion:

$$z(i,x) = \max\left\{\begin{array}{l} z(i-1,x); \\ z(i-1,x-l_i)+v_i | l_i \leqslant x, h_i \leqslant W; \\ z(i-1,x-h_i)+v_i | h_i \leqslant x, l_i \leqslant W; \end{array}\right\}$$

The recursion guarantees that each strip contains at most one copy of item i. It is based on the same idea as that of the recursion for 1D knapsack problem (Kellerer, Pferschy, & Pisinger, 2004). Define $q(i,x)$ as follows to record the solution path, $i = 1,\ldots,m$ and $x = 0,\ldots,L$: $q(i,x) = 0$ if $z(i,x) = z(i-1,x)$; $q(i,x) = 1$ if $z(i,x) = z(i-1,x-l_i)+v_i$; $q(i,x) = 2$ if $z(i,x) = z(i-1,x-h_i)+v_i$. The complexity for obtaining all the strips is $O(m^2L)$.

Let $K(j,x,i)$ be the number of item i in a type j strip with length x, where $K(j,x,i) \in \{0,1\}$, $j = 1,\ldots,2m$, $x = 0,\ldots,L$ and $i = 1,\ldots,m$. Let $f(j,x)$ be the value of strip $x \otimes W_j$ (length $\otimes$ width), $x = 0,\ldots,L$ and $j = 1,\ldots,2m$. For a particular j, $f(j,x) = z(m,x)$ can be determined for $x = 0,\ldots,L$ by solving the recursion with $W = W_j$. Meanwhile $q(i,x)$ is also obtained and can be used to determine $K(j,x,i)$ through backtracking, $i = 1,\ldots,m$, $x = 0,\ldots,L$.

3.1.3. Function GetPattern()

The following symbols are used to describe the pattern generation procedure.

$F(x,y)$	Value of bin $x \otimes y$
$N(x,y,i)$	Number of item i in bin $x \otimes y$
η_i	Number of item i in the current pattern
j	Index of the strip types, $j = 1,\ldots,2m$

The following recursion determines the value of bin $x \otimes y$:

$$F(x,y) = \max\left\{\begin{array}{l} \max[F(x,y-W_j)+U_{sp}(j,x) | W_j \leqslant y] \\ \max[F(x-W_j,y)+U_{sp}(j,y) | W_j \leqslant x] \end{array}\right\} \quad (1)$$

where

$$U_{sp}(j,x) = \sum_{i|K(j,x,i)>0} v_i[K(j,x,i) - N(x,y-W_j,i)] \quad (2)$$

$$U_{sp}(j,y) = \sum_{i|K(j,y,i)>0} v_i[K(j,y,i) - N(x-W_j,y,i)] \quad (3)$$

Recursion (1) indicates that for each strip type, two ways of placing it are considered. One is to place it horizontally along the upper edge of the sub-bin $x \otimes (y - W_j)$ and the other is to place it vertically along the right edge of the sub-bin $(x - W_j) \otimes y$.

The incremental value $U_{sp}(j,x)$ obtained from placing strip type j along the upper edge of sub-bin $x \otimes (y - W_j)$ is determined by formula (2). Let e_i be the effective frequency of item i used in computing the incremental value, then $U_{sp}(j,x) = \sum_{i=1}^{m} v_i e_i$. The effective frequency is determined as follows to guarantee that the frequency of each item is at most 1:

(1) $e_i = 0$ if $K(j,x,i) = 0$, because the item does not appear in the strip.
(2) $e_i = 1$ if $K(j,x,i) = 1$ and $N(x,y-W_j,i) = 0$. The item appears in the strip and not in the sub-bin.
(3) $e_i = 0$ if $K(j,x,i) = 1$ and $N(x,y-W_j,i) = 1$. The item appears in both the strip and sub-bin. The one in the strip is surplus and taken as waste.

Formula (3) can be similarly explained.

The function *GetPattern*() for generating each pattern (see *step 3* of the SVC2BPR) is based on recursion (1). Its contents are shown in the following box, where *continue* means that the next steps in the current loop are skipped and the next loop (related with the next value of the loop variable) should be started immediately; lines 6–7, 10, 11 and 13 indicate that the strips are skipped when:

(1) A smaller sub-bin has a higher value (lines 6–7).
(2) The current strip has value 0 or its width is too large to be packed into the current bin (line 10).
(3) The value of the bin cannot be improved when considering the strip (line 11).
(4) The value of the bin cannot be improved when the surplus items are taken out of consideration (lines 12 and 13). Surplus items are generated when $K(j,x,i) + N(x,y,i) > 1$ for some i.

Contents of function GetPattern ().

```
1  Let F(x,y) = 0, x = 0,...,L, y = 0,...,H.
2  Let N(x,0,i) = 0 and N(0,y,i) = 0, x = 0,...,L, y = 0,...,H,
   i = 1,...,m.
3  Call GetStrip (max (L,H)) to generate the strips.
4  For x = 0 to L
5     For y = 0 to H
6        If x > 0 and y > 0 and F(x,y) ⩽ max
   {F(x − 1,y),F(x,y − 1)}
7           Let F(x,y) = max{F(x − 1,y), F(x,y − 1)} and continue.
8  // Consider placing a horizontal strip along the upper edge of
   the sub-bin
9        For j = 1 to 2m
10          If f(j,x) = 0 or y + W_j > H then continue.
11          If F(x,y) + f(j,x) ⩽ F(x + W_j,y) then continue.
12          Let η_i = min(1,K(j,x,i) + N(x,y,i)), i = 1,...,m.
13          If Σ_{i=1}^{m} η_i v_i ⩽ F(x + W_j,y) then continue.
14          Let F(x + W_j,y) = Σ_{i=1}^{m} η_i v_i and N(x + W_j,y,i) = η_i,
   i = 1,...,m.
15 // Similarly consider placing a vertical strip along the right
   edge of the sub-bin.
```

3.2. Pattern-generation function for free patterns

A greedy procedure is used to generate free patterns. It considers discrete positions to place the items. Given the current

candidate positions and unpacked items, the procedure determines the item/position combination by a fitness level, places the selected item at the selected position, and updates the candidate positions and unpacked items. The process is repeated until one of the following cases appears: (1) all items have been packed; and (2) any unpacked item cannot be fitted into the bin.

At the beginning, the first item is packed with its left bottom corner coinciding with that of the bin. After that, a new item must be placed at a position where any packed item is below or to the left of it. The region where the remaining items can be packed in is called the envelope. The point where the slope of the envelope changes from vertical to horizontal is called a candidate position, at which items can be placed (that is, the left-bottom corner of an item must be placed at a candidate position). The slope of the envelope should be kept stair-shaped and go downward from left to right. An example is shown in Fig. 3 where the six items named 1–6 are packed into the bin one by one and the set of candidate positions is $P = \{p_1, p_2, p_3, p_4\}$. Envelops of such feature are also used in Wei, Zhang, and Chen (2009) and Martello, Pisinger, and Vigo (2000).

When a new item is placed at one of the candidate position, the set of the candidate positions and the shape of the envelope should be updated. Fig. 4 shows an example of the changes of the envelope and candidate positions after a new item is packed into the bin. Beginning from the envelope in Fig. 4(a), a new item is placed at position p2 and the new envelop is shown in Fig. 4(b). If the height of the region between the upper edge of the new item and the border of the bin (see the dark region in Fig. 4(c)) is less than the smallest edge-length of the remaining items, then none of the remaining items can be fitted into the region. Such regions are called bad regions. To reduce the complexity of the envelope, bad regions should be excluded from the envelope. The envelope in Fig. 4(c) is different from that in Fig. 4(b), because the bad region has been excluded. Fig. 5 shows another example for envelope update, where the bad region is along the right edge of the bin.

To decide which item to pack and which candidate position to place it at, a beam search method is used. Each combination of item and candidate position is a branch. By using a relatively simple strategy, we generate a pattern based on the current branch so as to obtain a lower bound for the value of the branch, where "value" denotes the total value of the items that are placed in the pattern. We assume that the branch with the largest lower bound has the greatest potential and should be selected.

Let $U_i = v_i/(l_i h_i)$ be the unit value of item i, $i = 1, \ldots, m$. Assume that the remaining items have been arranged according to the non-increasing order of their unit values. The following symbols are used to describe the pattern-generation procedure:

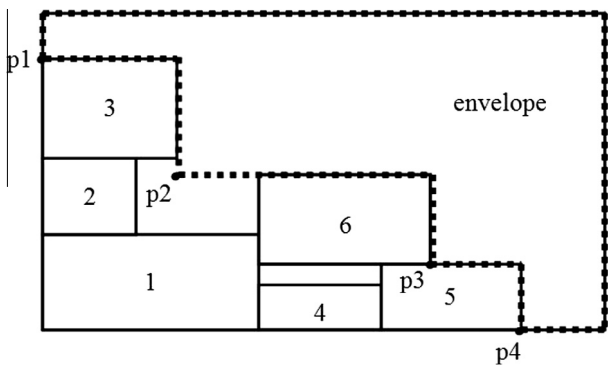

Fig. 3. Stair-shaped envelope.

Elop	Current envelope. Initially it contains only one candidate position (the left-bottom corner of the bin)
n_F	Number of candidate positions in *ELop*
V_{LB}	Lower bound of the current branch
maxLB	Maximum value among the lower bounds of all branches
optItem	ID of the item to pack
optPos	ID of the position to pack item *optItem*
optDir	Orientation of item *optItem*

Let GetLB $(I, J, itemDir, Elop)$ be the function that estimates the lower bound of the branch, in which item I is at position J with orientation *itemDir*(HORIZONTAL or VERTICAL). The contents of the pattern-generation procedure are shown in the following box. The current pattern is finished when $maxLB = 0$, because then either all items have been packed $(\sum_{i=1}^{m} b_i = 0)$ or none of the remaining items can be fitted into the bin.

Contents of the pattern-generation procedure.

```
While true
  Let maxLB = 0.
  For i = 1 to m
    Skip item i if b_i = 0.
    For itemDir ∈ {HORIZONTAL,VERTICAL}
      For j = 1 to n_F
        Let V_LB = GetLB(i,j,itemDir,Elop).
        If V_LB > maxLB then let maxLB = V_LB, optPos = j,
        optItem = i and optDir = itemDir.
  If maxLB = 0 then break.
  Put item optItem at position optPos in orientation optDir and
  update Elop.
  Let b_optItem = 0.
```

In the function GetLB $(I, J, itemDir, Elop)$, at first we put item I at position J with orientation *itemDir*, and then follow a maximum output first strategy to pack the remaining items. The remaining items are packed one by one according to the non-increasing order of their unit values, until no more items can be packed in. The pattern generated within this function is temporary (for obtaining a lower bound). It is dumped once the function ends.

The difference of the envelope's area before and after packing an item can be regarded as the equivalent area (A_e) the item occupied. Fig. 6 shows two examples of the equivalent area for the new item. When placing an item, according to the maximum output rule, the position with the minimum A_e should be selected.

When the current item (with length l and height h) is to be placed, the candidate positions on the slope are considered one by one to obtain the minimum A_e. If several positions lead to the same minimum A_e, the position that has the largest *GN* value is selected. Here *GN* denotes the *goodness number* (*GN*) that is determined as follows. Assume that the item is being placed at a candidate position formed by a horizontal line with length α and a vertical line with length β. If both $l = \alpha$ and $h = \beta$, *GN* of this pack is 2; if only one of them is equal, then $GN = 1$; otherwise $GN = 0$. An example for determining *GN* values is shown in Fig. 7.

The following symbols are used to describe *GetLB*$(I, J, itemDir, Elop)$:

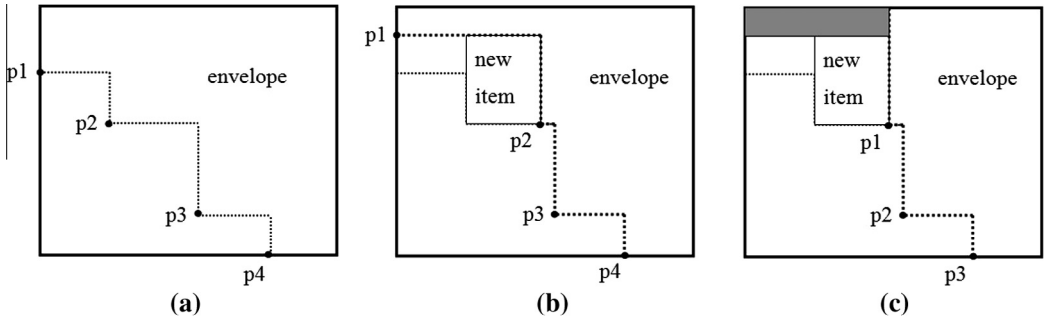

Fig. 4. Envelope update. (a) Original envelope. (b) Envelope before excluding the bad region. (c) Envelope after excluding the bad region.

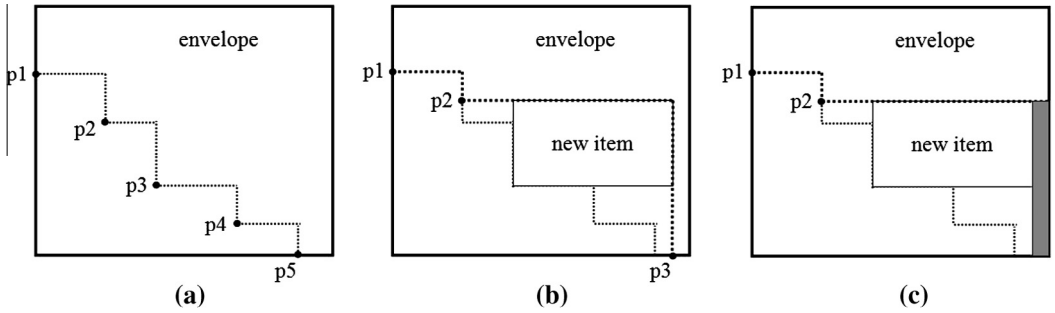

Fig. 5. Another example for envelope update. (a) Original envelope. (b) Envelope before excluding the bad region. (c) Envelope after excluding the bad region.

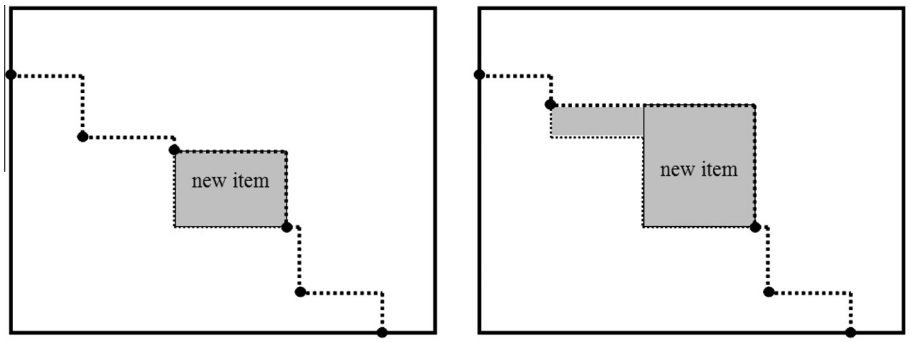

Fig. 6. Equivalent areas (dark regions).

ElopC	Current envelope retained within the function. Initially it is the same as *Elop*
n_C	Number of positions in *ELopC*
A_e^{ij}	Equivalent area when item i is packed at position j
GN_{ij}	*GN* value when item i is packed at position j
bestPos	ID of the position with the minimum equivalent area
bestEA	Minimum equivalent area
bestDir	Direction of the item related with *bestPos*
bestGN	*GN* value related with *bestPos*

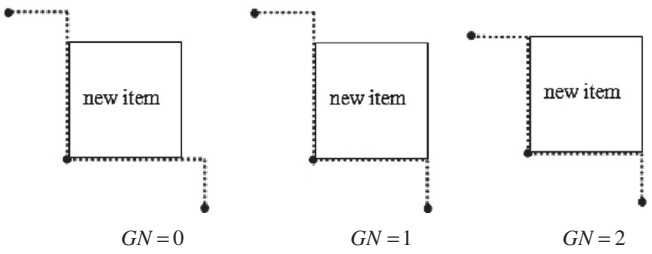

Fig. 7. Different *GN* values.

If an item with given orientation can be fitted into the bin at a candidate position, then the position is a feasible one. The contents of GetLB (*I*,*J*,*itemDir*,*Elop*) are shown in the following box. Lines 4–15 try to pack the remaining items one by one, until all items have been considered. Line 16 returns the lower bound to the host function *GetPattern*( ). For the current item under consideration, *bestEA* and *bestPos* are initialized in line 6. Lines 8–12 try to find the best position for the current item when it is placed in *itemDir*. Line 13 indicates that a feasible position has been found because *bestPos* > 0; then line 14 places the current item at the selected position and orientation; Line 15 updates the lower bound and the envelope.

Table 1
General performance of the algorithm.

Class	LB	CHBP	A-B	SVC2BPRG	Class	LB	CHBP	A-B	SVC2BPRG
1	6.6	**6.6**	**6.6**	**6.6**	6	1.0	**1.0**	**1.0**	**1.0**
	12.8	12.9	12.9	**12.8**		1.5	**1.8**	**1.8**	1.9
	19.5	**19.5**	**19.5**	**19.5**		2.1	**2.1**	**2.1**	**2.1**
	27.0	27.1	**27.0**	**27.0**		3.0	**3.0**	**3.0**	**3.0**
	31.3	**31.4**	**31.4**	**31.4**		3.2	**3.4**	**3.4**	**3.4**
	19.44	19.5	19.48	**19.46**		2.16	**2.26**	**2.26**	2.28
2	1.0	**1.0**	**1.0**	**1.0**	7	4.7	**5.2**	**5.2**	**5.2**
	1.9	**1.9**	**1.9**	2.0		9.9	10.4	**10.3**	**10.3**
	2.5	**2.5**	**2.5**	2.9		14.0	14.8	14.8	**14.5**
	3.1	**3.1**	**3.1**	3.3		20.0	21.1	21.1	**21.0**
	3.9	**3.9**	**3.9**	4.0		23.9	25.5	25.2	**25.1**
	2.48	**2.48**	**2.48**	2.64		14.5	15.4	15.32	**15.22**
3	4.7	4.8	4.9	**4.7**	8	5.0	**5.3**	**5.3**	**5.3**
	9.1	9.4	9.4	**9.2**		9.7	10.5	**10.4**	**10.4**
	13.2	13.6	13.7	**13.5**		14.2	15.0	15.0	**14.7**
	18.2	18.6	18.8	**18.4**		19.7	21.0	20.9	**20.7**
	21.5	**22.3**	**22.3**	**22.3**		24.2	25.8	25.6	**25.4**
	13.34	13.74	13.82	**13.62**		14.56	15.52	15.44	**15.30**
4	1.0	**1.0**	**1.0**	**1.0**	9	14.3	**14.3**	**14.3**	**14.3**
	1.9	**1.9**	**1.9**	**1.9**		27.5	**27.5**	**27.5**	**27.5**
	2.3	**2.5**	**2.5**	**2.5**		43.5	**43.5**	**43.5**	**43.5**
	3.0	3.3	**3.1**	**3.1**		57.3	**57.3**	**57.3**	**57.3**
	3.7	3.8	**3.7**	3.8		69.3	**69.3**	**69.3**	**69.3**
	2.38	2.5	**2.44**	2.46		42.38	**42.38**	**42.38**	**42.38**
5	5.9	**5.9**	6.0	**5.9**	10	3.9	**4.1**	**4.1**	**4.1**
	11.3	11.5	11.5	**11.4**		7.0	**7.3**	**7.3**	**7.3**
	17.1	17.6	17.6	**17.4**		9.5	**10.0**	**10.0**	**10.0**
	23.6	**24.0**	24.3	**24.0**		12.2	13.0	12.9	**12.6**
	27.2	28.2	28.4	**28.1**		15.3	**15.9**	16.1	**15.9**
	17.02	17.44	17.56	**17.36**		9.58	10.06	10.08	**9.98**
Total						13.784	14.128	14.132	14.070

Contents of *GetLB*(*I*,*J*,*itemDir*,*Elop*).

```
1  Return zero if position J is not feasible for placing item I in
   orientation itemDir.
2  Let ELopC = ELop. Place item I at position J and update
   ELopC.
3  Let V_LB = 0.
4  For i = 1 to m
5     Skip item i when b_i = 0.
6     Let bestEA = +∞ and bestPos = 0.
7     For itemDir ∈ {HORIZONTAL, VERTICAL}
8        For j = 1 to n_C
9           Skip position j if it is infeasible for placing the item in
   itemDir.
10          If A_e^ij ⩽ bestEA or (A_e^ij = bestEA and GN_ij > bestGN)
11             Let bestEA = A_e^ij and bestGN = GN_ij.
12             Let bestPos = j and bestDir = itemDir.
13    If bestPos > 0
14       Pack item i at position bestPos and in direction bestDir.
15       Let V_LB = V_LB + v_i and update ELopC.
16 Return V_LB.
```

3.3. Value correction

After each pattern is generated, the value correction procedure (see *step 5* of the SVC2BPR) corrects the values of the included items. The correction is based on the information of the current and previous patterns. The following formula is used:

$$v_i = \Omega v_i + \frac{(1-\Omega)[l_i h_i]^p}{u}$$

Table 2
Other results for SVC2BPRG.

Class	G_{best}	G_{total}	t_{best}	t_{total}
1	6.12	81.28	0.05	0.87
2	1.50	16.92	0.04	0.49
3	6.96	52.54	0.23	1.40
4	1.00	2.72	0.13	0.42
5	2.62	37.58	0.42	1.75
6	1.02	1.38	0.99	1.18
7	2.72	36.70	0.38	1.49
8	3.10	39.84	0.42	1.58
9	1.28	62.46	0.19	1.97
10	2.30	17.00	0.48	1.24
Average	2.86	34.84	0.33	1.24

where u denotes the material utilization of the current pattern, that is:

$$u = \frac{\sum_{i=1}^{m}(y_i l_i h_i)}{LH}$$

Ω is a real number in [0,1], p is a real number larger than 1, y_i denotes the number of type i item in the current pattern. The information of previous patterns is considered by the first term Ωv_i, and that of the current pattern by the second term. As seen in the formula, the value of an item increases as u decreases, meaning that the item does not combine well in the current pattern; therefore it needs to be used in priority in the pattern generation procedure. Similarly, larger items are also given priority, because they are difficult to pack.

The idea of value correction has been used in solving the one-dimensional cutting stock problem (Belov & Scheithauer, 2007) and the strip packing problem (Belov et al., 2008; Cui et al., 2013).

Table 3
Computational results.

	LB	IMA	HBP	ATS-BP	SVC2BPRF
Class 1	6.6	**6.6**	**6.6**	**6.6**	**6.6**
	12.8	12.9	12.9	12.9	**12.8**
	19.5	**19.5**	**19.5**	**19.5**	**19.5**
	27.0	**27.0**	**27.0**	**27.0**	**27.0**
	31.3	**31.3**	**31.3**	31.4	**31.3**
	19.44	19.46	19.46	19.48	**19.44**
Time		0.100	0.285	2.14	0.60
Class 2	1.0	**1.0**	**1.0**	**1.0**	**1.0**
	1.9	**1.9**	**1.9**	**1.9**	2.0
	2.5	**2.5**	**2.5**	**2.5**	**2.5**
	3.1	**3.1**	**3.1**	**3.1**	3.2
	3.9	**3.9**	**3.9**	**3.9**	4.0
	2.48	**2.48**	**2.48**	**2.48**	2.54
Time		0.009	0.128	5.92	0.32
Class 3	4.7	**4.7**	**4.7**	**4.7**	**4.7**
	9.1	9.4	9.4	9.4	**9.2**
	13.2	13.5	13.5	13.6	**13.4**
	18.2	**18.4**	**18.4**	18.6	**18.4**
	21.5	22.2	22.2	22.3	**22.0**
	13.34	13.64	13.64	13.72	**13.54**
Time		0.380	0.980	30.94	0.96
Class 4	1.0	**1.0**	**1.0**	**1.0**	**1.0**
	1.9	**1.9**	**1.9**	**1.9**	**1.9**
	2.3	2.5	2.5	**2.4**	2.5
	3.0	**3.1**	3.2	3.2	3.2
	3.7	**3.7**	3.8	3.8	3.9
	2.38	**2.44**	2.48	2.46	2.50
Time		0.158	2.107	14.91	0.43
Class 5	5.9	**5.9**	**5.9**	**5.9**	**5.9**
	11.3	**11.4**	11.5	**11.4**	**11.4**
	17.1	17.4	17.5	17.5	**17.3**
	23.6	**23.9**	24.0	**23.9**	**23.9**
	27.2	27.9	28.0	28.0	**27.7**
	17.02	17.3	17.38	17.34	**17.24**
Time		0.357	1.968	91.27	1.03
Class 6	1.0	**1.0**	**1.0**	**1.0**	**1.0**
	1.5	1.7	1.7	**1.6**	1.7
	2.1	**2.1**	**2.1**	**2.1**	**2.1**
	3.0	**3.0**	**3.0**	**3.0**	**3.0**
	3.2	**3.2**	3.4	3.4	3.4
	2.16	**2.20**	2.24	2.22	2.24
Time		0.125	2.998	8.02	0.42
Class 7	4.7	**5.2**	**5.2**	**5.2**	**5.2**
	9.9	10.4	10.5	10.4	**10.2**
	14.0	14.7	15.1	14.6	**14.5**
	20.0	21.2	21.8	21.3	**20.8**
	23.9	25.3	25.9	25.5	**25.0**
	14.56	15.44	15.76	15.52	**15.14**
Time		0.779	3.571	241.09	0.97
Class 8	5.0	**5.3**	**5.3**	**5.3**	**5.3**
	9.7	10.4	10.5	10.4	**10.3**
	14.2	15.0	15.4	15.1	**14.7**
	19.7	20.8	21.3	20.8	**20.5**
	24.2	25.7	26.3	26.0	**25.3**
	14.56	15.44	15.76	15.52	**15.22**
Time		0.794	3.642	178.65	0.93
Class 9	14.3	**14.3**	**14.3**	**14.3**	**14.3**
	27.5	**27.5**	**27.5**	27.6	**27.5**
	43.5	**43.5**	**43.5**	**43.5**	**43.5**
	57.3	**57.3**	**57.3**	**57.3**	**57.3**
	69.3	**69.3**	**69.3**	**69.3**	**69.3**
	42.38	**42.38**	**42.38**	42.4	**42.38**
Time		0.000	0.250	11.68	0.91
Class 10	3.9	**4.1**	**4.1**	**4.1**	**4.1**
	7.0	7.3	7.3	7.3	**7.2**
	9.5	10.1	10.0	**9.9**	**9.9**
	12.2	12.8	12.8	12.8	**12.6**
	15.3	15.8	16.0	15.9	**15.5**
	9.58	10.02	10.04	10.00	**9.86**
Time		0.484	2.313	117.48	0.55
Total bins		14.072	14.156	14.102	**14.01**
Avg. time		0.319	1.824	70.21	0.71

Table 4
Other results for SVC2BPRF.

Class	G_{best}	G_{total}	t_{best}	t_{total}
1	1.22	97.22	0.01	0.60
2	1.22	2.68	0.13	0.32
3	3.10	106.04	0.03	0.96
4	1.04	2.44	0.13	0.43
5	7.08	155.92	0.06	1.03
6	1.20	2.54	0.19	0.42
7	8.60	127.88	0.06	0.97
8	7.66	130.58	0.05	0.93
9	1.10	193.66	0.01	0.91
10	2.18	59.40	0.04	0.55
Average	3.44	87.84	0.07	0.71

Table 5
Solution value related with different p values.

Class	$p = 1.03$	$p = 1.05$	$p = 1.07$
1	972	973	973
2	131	132	132
3	684	681	679
4	125	123	125
5	871	868	867
6	115	114	115
7	764	761	761
8	767	765	765
9	2119	2119	2119
10	499	499	495
Total	7047	7035	7031

Table 6
Solution value related with different Ω values.

Class	$\Omega = 0.3$	$\Omega = 0.5$	$\Omega = 0.7$
1	972	973	973
2	133	132	132
3	681	681	683
4	125	123	123
5	867	868	869
6	115	114	115
7	760	761	763
8	765	765	767
9	2119	2119	2119
10	496	499	495
Total	7033	7035	7039

4. Computational results

The algorithm SVC2BPR was coded in C++ and executed on a PC (Intel Core2 Quad CPU Q9400 2.66 gigahertz, RAM 4 gigabyte). The following default parameter values are used in Sections 4.1 and 4.2: $\Omega = 0.5$, $p = 1.05$, $G_{max} = 200$ and $t_{max} = 2$ seconds (maximum computation time for an instance). A test on parameter sensitivity is available in Section 4.3.

4.1. General test on guillotine patterns

In this sub-section, the SVC2BPRG is compared with two published algorithms for 2BPRG. One is the agent-based algorithm (referred to as the A-B) in Polyakovsky and M'Hallah (2009) and the other is that (referred to as the CHBP) in Charalambous and Fleszer (2011). These two algorihms are selected because the computational results in Fleszar (2013) indicate that they outperform other algorithms in solving 2BPRG instances. Ten classes of benchmark instances from Berkey and Wang (1987) (classes 1–6) and Lodi et al. (1999) (classes 7–10) are used.

Table 7
Solution value related with different G_{max} values.

Class	G_{max} = 1	G_{max} = 3	G_{max} = 5	G_{max} = 10	G_{max} = 25	G_{max} = 50	G_{max} = 200
1	1015	984	980	976	975	975	973
2	137	136	135	134	133	133	132
3	754	701	693	687	686	684	681
4	125	125	125	125	125	125	123
5	939	878	871	868	868	868	868
6	115	114	114	114	114	114	114
7	801	769	763	763	761	761	761
8	805	777	770	766	764	765	765
9	2131	2119	2119	2119	2119	2119	2119
10	534	504	498	497	496	496	499
Total	7356	7107	7068	7049	7041	7040	7035

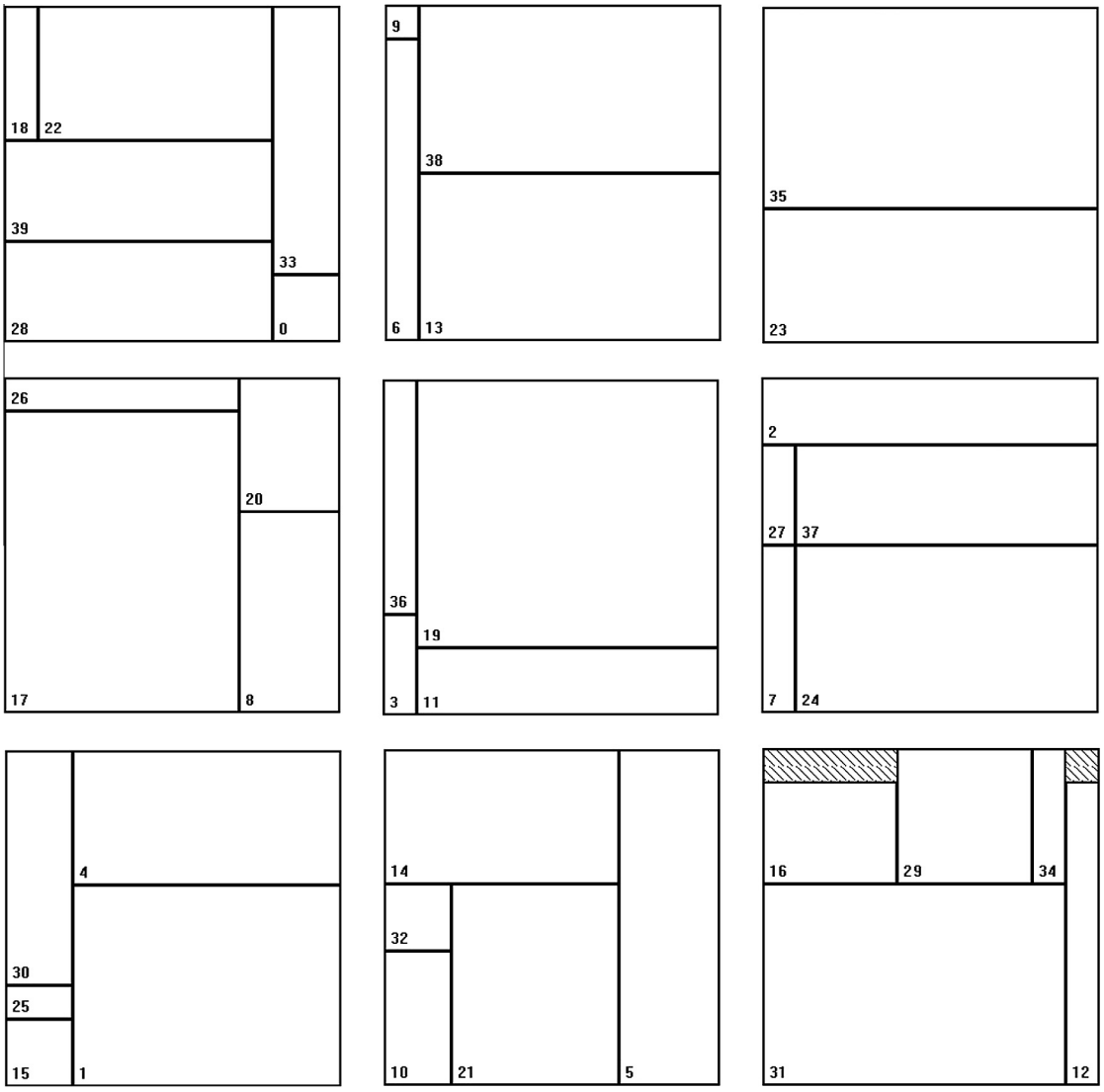

Fig. 8. Solution to the example (9 bins).

In classes 1–6, the data are generated using the following rules:

Class 1 $L = H = 10$, l_i and h_i uniformly in [1,10].
Class 2 $L = H = 30$, l_i and h_i uniformly in [1,10].
Class 3 $L = H = 40$, l_i and h_i uniformly in [1,35].
Class 4 $L = H = 100$, l_i and h_i uniformly in [1,35].
Class 5 $L = H = 100$, l_i and h_i uniformly in [1,100].
Class 6 $L = H = 300$, l_i and h_i uniformly in [1,100].

There are four different item types in classes 7–10:

(1) l_i uniformly in $[1,1/2L]$ and h_i uniformly in $[2/3H, H]$.
(2) l_i uniformly in $[2/3L,L]$ and h_i uniformly in $[1,1/2H]$.
(3) l_i uniformly in $[1/2L,L]$ and h_i uniformly in $[1/2H,H]$.
(4) l_i uniformly in $[1,1/2L]$ and h_i uniformly in $[1,1/2H]$.

The items in classes 7–10 are generated as following and the bin sizes are $W = L = 100$:

Class 7	70% of type 1 items and 10% each for types 2, 3 and 4.
Class 8	70% of type 2 items and 10% each for types 1, 3 and 4.
Class 9	70% of type 3 items and 10% each for types 1, 2 and 4.
Class 10	70% of type 4 items and 10% each for types 1, 2 and 3.

Each class contains 5 groups and each group includes 10 instances with the same number of items (m).

Table 1 shows the average solution value (number of bins) of an instance for all groups, where the results for each class occupy a block. The average solution value of an instance for the class is shown in the last line of the block. The lower bounds (LB) presented in Clautiaux et al. (2007) are also listed for comparison.

The lowest solution value of each group obtained by the three algorithms is regarded as the best solution (denoted in bold number in the table). The number of groups in which the SVC2BPRG achieves the best solution is 44 (out of 50). It is 32 for the A-B and 30 for the CHBP. The overall average solution value (counted over all groups) obtained by each algorithm is given in the last line of Table 1. The SVC2BPRG yields the smallest average solution value (14.07), followed by the CHBP (14.128) and A-B (14.132). The results indicate that the SVC2BPRG is the most efficient in improving solution quality. The detailed SVC2BPRG solution for the first instance in Group 2 of Class 1 is given in Appendix.

Table 2 shows the other results of the SVC2BPRG, where G_{best} is the number of solutions generated when the best solution is obtained (averaged over all instances in the class), G_{total} is the total number of solutions generated, t_{best} and t_{total} denote the corresponding computation times (in seconds). For a particular instance, the G_{best} value is in [1,174] and the G_{total} value in [1,200]. The average G_{best} value is 2.86, indicating that the best solutions can often be obtained in the early stage of the solution process.

The average computation time of an instance is 1.24 seconds for SVC2BPRG. It is less than 3.5 seconds for A-B (on a computer with processor Athlon XP 2800) and equal to 0.066 seconds for CHBP (on a computer with processor core i3, 2.13 gigahertz). The computation times may all be seen as reasonable.

4.2. General test on free patterns

In this section, algorithm SVC2BPRF is compared with three published algorithms for 2BPRF: the IMA (Hayek et al., 2008), HBP (Boschetti & Mingozzi, 2003) and ATS-BP (Harwig et al., 2006), using the benchmark instances in Section 4.1.

Table 3 shows the average solution value (number of bins) of an instance for all groups, where the results for each class occupy a block. The computational results of the other three algorithms are obtained from Hayek et al. (2008). The average solution value and the average computation time of an instance for the class are shown in the last two lines of the block.

The lowest solution value of each group obtained by the four algorithms is regarded as the best solution (denoted in bold number in the table). The number of groups in which the SVC2BPRF achieves the best solution is 42 (out of 50). It is 30 for the IMA, 25 for the HBP and 27 for the ATS-BP. The overall average solution value (counted over all 50 groups) obtained by each algorithm is given in the last row of Table 3. The SVC2BPRF yields the smallest average solution value (14.01), followed by IMA (14.072), ATS-BP (14.102) and HBP (14.156). The results indicate that the SVC2BPRF is the most efficient in improving solution quality.

Table 8
Item dimensions.

Id	Dimension	Id	Dimension	Id	Dimension	Id	Dimension
0	2×2	10	2×4	20	3×4	30	2×7
1	8×6	11	2×9	21	5×6	31	9×6
2	2×10	12	9×1	22	7×4	32	2×2
3	3×1	13	5×9	23	4×10	33	8×2
4	4×8	14	7×4	24	5×9	34	1×4
5	10×3	15	2×2	25	2×1	35	6×10
6	9×1	16	4×3	26	1×7	36	1×7
7	5×1	17	7×9	27	1×3	37	9×3
8	3×6	18	1×4	28	3×8	38	5×9
9	1×1	19	8×9	29	4×4	39	8×3

Table 4 shows the other results of the SVC2BPRF. For a particular instance, the G_{best} value is in [1,157] and the G_{total} value in [1,200]. The average G_{best} value is 3.44, indicating that the best solutions can often be obtained in the early stage of the solution process.

The average computation time of an instance is 0.71 seconds for SVC2BPRF, 0.319 seconds for IMA (Intel Pentium 4 2.66 gigahertz), 1.824 seconds for HBP (Intel Pentium 3 933 megahertz), and 70.21 seconds for ATS-BP (unknown CPU). The computation time of the SVC2BPRF may be seen as reasonable.

4.3. Parameter sensitivity

In this section, the sensitivity of the SVC2BPRG to the parameters p, Ω and $G_{\max}$ is tested. Recall that the default values are $\Omega = 0.5$, $p = 1.05$, $G_{\max} = 200$. In testing the impact of a parameter on the quality of the solutions, this parameter is allowed to vary and the other parameters assume the default values.

Table 5 shows the impact of the p value. The total number of bins (sum of the solution values of all instances) related with each p value is listed in the last row of the table. It varies from 7031 to 7047. This indicates that the solution quality is slightly sensitive (not serious) to the p value when $p \in [1.03, 1.07]$.

Table 6 shows the impact of the Ω value. The total number of bins is between 7033 and 7039, indicating that the solution quality is slightly sensitive (not serious) to the Ω value when $\Omega \in [0.3, 0.7]$.

Table 7 shows the impact of the $G_{\max}$ value. The total number of bins is between 7035 and 7356. It is seen that the solution quality is sensitive to $G_{\max}$ when $G_{\max} < 25$, and not sensitive when $G_{\max} \geqslant 25$.

5. Conclusions

The sequential value correction heuristic SVC2BPR for the 2BPR is proposed in this paper. It is rely on the combination of pattern generation procedures and the value correction scheme. The computational results indicate that the algorithm is more efficient in improving solution quality than five published algorithms for the instances tested.

Future research direction may include the extending of the algorithm to solve the 2BPR with multiple bins and the two dimensional cutting stock problem.

Acknowledgements

This research is part of Projects 61363026 and 71371058 supported by National Natural Science Foundation of China. It is also supported by the scientific research project 201010LX003 of the universities in Guangxi.

Appendix A. Solution example

The 2BPRG solution for the first instance in group 2 of class 1 is given in Fig. 8. The bin has a dimension of 10×10. The dimensions of the items are listed in Table 8. The solution is optimal because the number of bins is equal to the continuous lower bound $\lceil \sum_{i=1}^{m} l_i h_i / (LH) \rceil$.

Appendix B. Supplementary material

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.ejor.2014.06.032.

References

Belov, G., & Scheithauer, G. (2007). Setup and open-stacks minimization in one-dimensional stock cutting. *INFORMS Journal on Computing, 19*, 27–35.

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.

Bengtsson, B. (1982). Packing rectangular pieces – A heuristic approach. *The Computer Journal, 25*, 353–357.

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

Boschetti, M. A., & Mingozzi, A. (2003). The two-dimensional finite bin-packing problem. Part II: New lower and upper bounds. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies, 1*, 135–147.

Chan, T. M., Alvelos, F., Silva, E., & Carvalho, J. (2011). Heuristics with stochastic neighborhood structures for two-dimensional bin packing and cutting stock problems. *Asia-Pacific Journal of Operational Research, 28*, 255–278.

Charalambous, C., & Fleszer, K. (2011). A constructive bin-oriented heuristic for the two-dimensional bin packing problem with guillotine cuts. *Computers & Operations Research, 38*, 1443–1451.

Clautiaux, F., Jouglet, A., & Hayek, E. J. (2007). A new lower bound for the non-oriented two-dimensional bin-packing problem. *Operations Research Letters, 35*, 365–373.

Cui, Y., Yang, L., & Chen, Q. (2013). Heuristic for the rectangular strip packing problem with rotation of items. *Computers & Operations Research, 40*, 1094–1099.

Cui, Y., & Zhang, X. (2007). Two-stage general block patterns for the two-dimensional cutting problem. *Computer & Operations Research, 34*, 2882–2893.

Dell'Amico, M., Martello, S., & Vigo, D. (2002). A lower bound for the non-oriented two-dimensional bin packing problem. *Discrete Applied Mathematics, 118*, 13–24.

Fleszar, K. (2013). Three insertion heuristics and a justification improvement heuristic for two-dimensional bin packing with guillotine cuts. *Computer & Operations Research, 40*, 463–474.

G, Y-G., Seong, Y-J., & Kang, M-K. (2003). A best-first branch and bound algorithm for unconstrained two-dimensional cutting problems. *Operations Research Letters, 31*, 301–307.

Han, W., Bennell, J. A., & Zhao, X. (2013). Construction heuristics for two-dimensional irregular shape bin packing with guillotine constraints. *European Journal of Operational Research, 230*, 495–504.

Harwig, J. M., Barnes, J. W., & James, T. (2006). An adaptive tabu search approach for 2-dimensional orthogonal packing problems. *Military Operations Research, 11*, 5–26.

Hayek, J. E., Moukrim, A., & Negre, S. (2008). New resolution algorithm and pretreatments for the two-dimensional bin-packing problem. *Computers & Operations Research, 35*, 3184–3201.

Kellerer, H., Pferschy, U., & Pisinger, D. (2004). *Knapsack Problems*. Berlin and Heidelberg GmbH & Co.K: Springer-Verlag (Chapter 7).

Liao, C-S., & Hsu, C-H. (2013). New lower bounds for the three-dimensional orthogonal bin packing problem. *European Journal of Operational Research, 225*, 244–252.

Lodi, A., Martello, S., & Daniele, V. (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., & Vigo, D. (2002A). Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics, 123*, 379–396.

Lodi, A., Martello, S., & Vigo, D. (2002B). Two-dimensional packing problems: a survey. *European Journal of Operational Research, 141*, 241–252.

Mack, D., & Bortfelt, A. (2012). A heuristic for solving large bin packing problems in two and three dimensions. *Central European Journal of Operational Research, 20*, 337–354.

Martello, S., Pisinger, D., & Vigo, D. (2000). The three-dimensional bin packing problem. *Operations Research, 48*, 256–267.

Polyakovsky, S., & M'Hallah, R. (2009). An agent-based approach to the two-dimensional guillotine bin packing problem. *European Journal of Operational Research, 192*, 767–781.

Wei, L., Oon, W-C., & Zhu, W. (2013). A goal-driven approach to the 2D bin packing and variable-sized bin packing problems. *European Journal of Operational Research, 224*, 110–121.

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