European Journal of Operational Research 238 (2014) 95–103

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

# A hybrid heuristic algorithm for the 2D variable-sized bin packing problem

Shaohui Hong [a], Defu Zhang [a,*], Hoong Chuin Lau [b], XiangXiang Zeng [a], Yain-Whar Si [c]

[a] Department of Computer Science, Xiamen University, Xiamen 361005, China
[b] School of Information Systems, Singapore Management University, Singapore
[c] Department of Computer and Information Science, University of Macau, Macau

ARTICLE INFO

*Article history:*
Received 26 July 2013
Accepted 31 March 2014
Available online 12 April 2014

*Keywords:*
Packing
Guillotine constraint
Simulated annealing
Heuristic

ABSTRACT

In this paper, we consider the two-dimensional variable-sized bin packing problem (2DVSBPP) with guillotine constraint. 2DVSBPP is a well-known NP-hard optimization problem which has several real applications. A mixed bin packing algorithm (MixPacking) which combines a heuristic packing algorithm with the Best Fit algorithm is proposed to solve the single bin problem, and then a backtracking algorithm which embeds MixPacking is developed to solve the 2DVSBPP. A hybrid heuristic algorithm based on iterative simulated annealing and binary search (named HHA) is then developed to further improve the results of our Backtracking algorithm. Computational experiments on the benchmark instances for 2DVSBPP show that HHA has achieved good results and outperforms existing algorithms.



## 1. Introduction

Combinatorial optimization plays a very important role in operational research, discrete mathematics and computer science. The two-dimensional bin packing problem (2DBPP) is a well-known combinatorial optimization problem. There are many variants of 2DBPP, one of which is the variable-sized bin packing problem (2DVSBPP) which is a generalization of the standard 2DBPP. In 2DVSBPP, we are given a set of bins, each bin $j$ with dimensions $W_j \times H_j$, $j \in [1,n]$ having a type represented by a finite or infinite number $N_j$, and a set of items each item $i$ with dimensions $w_i \times h_i$, $i \in [1,n]$, the objective is to use minimum bins (more precisely, minimum total area of used bins) to pack all items that meet the following constraints (Lodi, Martello, & Vigo, 1999a).

**Orientation**: Each item may either have a fixed orientation or can be rotated.
**Guillotine cut**: It may or may not to be imposed that the items are obtained through a sequence of edge-to-edge cut parallel to the edges of the respective bin.

2DVSBPP with guillotine cuts constraints is abbreviated 2DVSBPP|O|G. 2DBPP is known to be an NP-hard problem, 2DVSBPP|O|G is more computationally challenging than 2DBPP. This is the reason why several approximate methods have been proposed to solve 2DBPP, which are generally based on heuristics and meta-heuristics (Lodi et al., 1999a). Some heuristic algorithms such as the first-fit decreasing height (FFDH) algorithm (Coffman, Edward, Garey, Johnson, & Tarjan, 1980), the best-fit decreasing height (BFDH) algorithm (Coffman & Shor, 1990), the floor-ceiling (FC) algorithm (Lodi, Martello, & Vigo, 1999b; Lodi et al., 1999a), improved best-fit decreasing height (BFDH*) algorithm (Bortfeldt, 2006), JOIN (Martello, Monaci, & Vigo, 2003), size-alternating stack (SAS) algorithm (Ntene, 2007; Ntene & van Vuuren, 2009), modified size-alternating stack (SASm) algorithm (Ortmann, Ntene, & van Vuuren, 2010) and best-fit with stacking (BFS) algorithm (Ortmann et al., 2010) for two-dimensional strip packing problem were developed for 2DBPP. A two-phase approach was applied to the FC algorithm resulting in the hybrid floor-ceiling (HFC) algorithm for 2DBPP (Lodi et al., 1999a, 1999b). Lodi, Martello, and Vigo (2002) reviewed recent advances on solving 2DBPP. Other methods include the work of Pisinger and Sigurd (2007) based on decomposition techniques and constraint programming. Most recently, Omar and Ramakrishnan (2013) proposed an evolutionary particle swarm optimization algorithm for solving non-oriented 2DBPP.

The literature mentioned above focus on two-dimensional bin packing with bins of the same size. For 2DVSBPP, Valério de Carvalho (2002) defined an LP formulation. Kang and Park (2003) proposed two greedy algorithms with a worst-case performance bound of 3/2. Pisinger and Sigurd (2005) used a branch-and-price

* Corresponding author. Tel.: +86 1895 9217108; fax: +86 0592 2580258.
*E-mail address:* dfzhang@xmu.edu.cn (D. Zhang).

http://dx.doi.org/10.1016/j.ejor.2014.03.049
0377-2217/

algorithm to solve the problem exactly. Liu, Chu, and Wang (2011) developed a dynamic programming-based heuristic for 2DVSBPP. Recently, Wei, Oon, Zhu, and Lim (2013) proposed a goal-driven approach (GDA) for 2DBPP and 2DVSBPP, and achieved the best published results in these two problems. Alvarez-Valdes, Parreño, and Tamarit (2013) presented a GRASP/Path relinking algorithm for two- and three-dimensional multiple bin-size bin packing problems.

There are a few papers on 2DBPP with guillotine cuts constraints. Polyakovsky and M'Hallah (2009) studied the two-dimensional guillotine bin packing problem using a new Guillotine Bottom Left (GBL) constructive heuristic and its Agent-Based (A-B) implementation. A-B is particularly fast and yields near-optimal solutions. Ortmann et al. (2010) proposed a two-stage algorithm for 2DVSBPP described as follows. In the first stage, sort bins by DAIP (decreasing area increasing perimeter), and sort items by DHDW (decreasing height decreasing width) and use a specified strip algorithm to pack items into a strip with the strip width as the width of the first bin in the list. Repeat the previous phase until there are no unpacked items. In the second stage, for each packed bin, try to use the specified strip algorithm to pack all items in the packed bin into a bin with a smaller area and not used, if such smaller bin exists, then replace the pack on the packed bin and free it. Charalambous and Fleszar (2011) developed a constructive bin-oriented heuristic for the two-dimensional bin packing problem with guillotine cuts. Recently, three insertion heuristics and a justification improvement heuristic were developed for 2DBPP with guillotine cuts (Fleszar, 2013).

The literature on 2DVSBPP|O|G is relatively scarce, up to now, the publications to our knowledge include Ortmann et al. (2010). In this paper, we consider 2DVSBPP|O|G and develop a hybrid heuristic algorithm. The paper is organized as follows. In the Section 2, we introduce a heuristic named BTVS for 2DVSBPP. Section 3 presents a hybrid heuristic algorithm named HHA to continuously improve the solution of BTVS. In Section 4, various benchmark problems are used to examine the effectiveness of HHA. At last, Section 5 provides concluding remarks.

## 2. Backtracking algorithm

### 2.1. Heuristic packing algorithm (HPA)

The constructive heuristic algorithm (Leung, Zhang, & Sim, 2011) uses a score strategy to evaluate each item to be packed for an available space, and then select one item with highest score to pack that space. The score strategy has been shown in Leung et al. (2011) to be very effective for solving strip packing. Using the same technique, we consider all available spaces, and then evaluate the current item for each available space, then select one space with the highest score to pack the current item. Although the constructive algorithm by Leung et al. (2011) evaluates each item and each available space, the proposed procedure HPA uses an ordered list of items and considers one item at a time. Usually the piece is placed on the left bottom corner of the space, such as Figs. 1 and 2. More precisely, our score strategy is shown in Table 1.

Column one of Table 1 gives the various conditions that can occur. The next column Fitness is the score to evaluate which space to place the item, the space with highest nonnegative fitness would be taken. The third column shows the corresponding change of in the space value of the candidate if that space had been used to place the item. Specifically for condition row 1, it is assigned the highest fitness because the item will fully use the space and the space would no longer be split, thus the m value is set to $-1$. For the second row, one edge length of the space is equal to the item's length, so it will fully use either the left or the bottom part of the space, and the remainder space will be a new space so the $m$ value is set to zero. For the third row, it will generate a sub space on top and a sub space on right, hence its $m$ value is set to 1. For the fourth row, the space cannot cover the item so it is assigned a negative value, hence it will not change the space and its m value is set to 0.

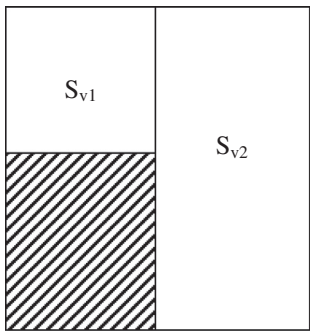


**Fig. 1.** Vertical pattern.

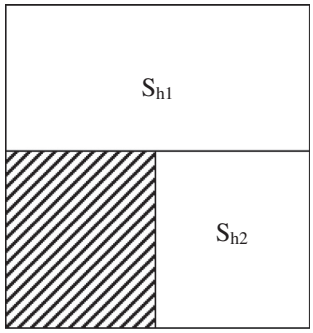


**Fig. 2.** Horizontal pattern.

**Table 1**
Score strategy.

| Condition (space dimension = $w \times h$, item dimension = $w_i \times h_i$) | Fitness | $m$ (change of space number in candidate) |
|---|---|---|
| $w = w_i$ and $h = h_i$ | 3 | $-1$ |
| ($w = w_i$ and $h > h_i$) or ($w > w_i$ and $h = h_i$) | 1 | 0 |
| $w > w_i$ and $h > h_i$ | 0 | 1 |
| $w < w_i$ or $h < h_i$ | $-1$ | 0 |

After choosing the best space to place the current item, we need to cut the sub space from best space which meets the guillotine constraint. In general, we have two cut patterns as shown in Figs. 1 and 2, where the current item to be placed is shown in shade. The vertical pattern first cuts from top to bottom along the current item's width, it will generate two new sub spaces $S_{v1}$ and $S_{v2}$.The horizontal pattern cuts from left to right along the current item's height, it will generate two new sub spaces $s$ $S_{h1}$ and $S_{h2}$. Both patterns meet the guillotine constraint.

Since we cannot determine which cut is better, we use a mutex-space (MutexSpace) to record these two patterns at the same time. When choosing a best space for an item, mutex-space records the space, item and the four new sub spaces ($S_{v1}$ and $S_{v2}$ in vertical pattern, $S_{h1}$ and $S_{h2}$ in horizontal pattern), where $S_{v1}$ and $S_{v2}$ are incompatible with (mutex) $S_{h1}$ and $S_{h2}$. During the search, if the best space BestSpace is found for the current item, we remove the best space and its mutex-spaces from space list (SL). At the same time, we add new subspaces that are generated by packing the current item (ITEM) in the two patterns. As Figs. 1 and 2 show, they are two different patterns for the same space and same item.

For example, we use a space SPACE1 to place an item ITEM1, we store both patterns and add $S_{v1}$, $S_{v2}$, $S_{h1}$, $S_{h2}$ to the space list. When we process an item ITEM2 and find that $S_{h1}$ is the best space for ITEM2, we need to apply the horizontal pattern to SPACE1 and ITEM1. And as we apply the horizontal pattern, there is no possible to generate $S_{v1}$ and $S_{v2}$, so we remove $S_{v1}$ and $S_{v2}$ from space list. Then we retain $S_{h2}$ in space list, and remove $S_{h1}$ from space list and process $S_{h1}$ and ITEM2 in the phases at the start of this paragraph.

The heuristic packing algorithm is presented as Algorithm 1 in detail and named as HPA. The input of HPA is a bin (BIN) and a list of items (ITEMS), the output of HPA is a list of unpacked items and a list of placements. UP is short for unpacked list. If the item does not fit, the item was added into the unpacked list UP.

### 2.2. Best Fit packing algorithm (BFA)

The Best Fit with Stacking algorithm (BFS) proposed by Ortmann et al. (2010) improves the BFDH* algorithm presented by Bortfeldt (2006). When packing an item at one level, it may generate a new sub space at the ceiling of the space due to the guillotine constraint, BFS uses an FFDH (first-fit decreasing height) strategy to pack the unpacked items into the ceiling sub space when it is generated.

**Algorithm 1**. Heuristic packing algorithm (HPA)

**Input**: a bin BIN, a list of items ITEMS
**Output**: a list of placement on bin, a list of unpacked items
HPA(BIN, ITEMS)
1: initialize a list of space SL (short for space list) and add BIN into SL
2: initialize an empty list UP (short for unpacked list) of unpacked items
3: initialize an empty placement list PL (short for placement list)
4: **for** ITEM **in** ITEMS
5: find a space SPACE with max score(SPACE, ITEM) in space list SL
6: **if** cannot find such SPACE **then**
7: add ITEM into UP
8: continue
9: remove SPACE from spaces list SL
10: add placement(SPACE, ITEM) in left bottom position into placement list PL
11: **if** SPACE is a sub-space of mutex-space MutexSpace **then**
12: apply the cut pattern of SPACE to MutexSpace
13: remove other two sub spaces in other pattern from space list SL
14: add another sub space of the applied pattern into space list SL
15: add sub spaces of mutex-space(SPACE, ITEM) into space list SL
16: **return** PL, UP

BFS was proposed to solve strip packing problem, which sorts items with decreasing height order, and packs them in horizontal direction. In this paper, we apply this algorithm to the bin packing problem and consider the vertical direction as well. The Best Fit packing algorithm (BFA) proposed in this paper is developed from BFS, and its detailed description is given in Algorithm 2. BFA regards this sub space as a new space having the same priority to other spaces on the space list, and only uses it to pack an item when it has the highest score, which is the major difference compared with the BFS.

**Algorithm 2**. Best Fit packing algorithm (BFA)

**Input**: place pattern PATTERN, a bin BIN, A list of items ITEMS
**Output**: a list of placement on bin, a list of unpacked items
BFA(PATTERN, BIN, ITEMS)
1: sort the items by DHDW (or DWDH)
2: initialize a list of space SL and add BIN into SL
3: initialize an empty list UP of unpacked items
4: initialize an empty placement list PL
5: **for** ITEM **in** ITEMS
6: find the space SPACE with minimum horizontal(or vertical) residual space for item ITEM
7: **if** no space fit for ITEM **then**
8: add ITEM into UP
9: **continue**
10: add placement(SPACE, ITEM) in left bottom position into placement list PL
11: cut space SPACE with item ITEM in horizontal(or vertical) pattern and add the two sub spaces generated into SL
12: **return** PL, UP

The input of BFA is a bin, a list of item and a place pattern (PATTERN). The place pattern has two patterns: horizontal pattern (Fig. 2) or vertical pattern (Fig. 1). Algorithm MixPacking (will be described in Section 2.3) invokes BFA with a different PATTERN, where the horizontal PATTERN uses DHDW (decreasing height decreasing width), and the vertical PATTERN uses DWDH (decreasing width decreasing height) in the first line of BFA.

### 2.3. Mixed bin packing algorithm (MixPacking)

Before invoking algorithm HPA, we need to sort items in a specify order. Since it is hard to say which order is better for a special bin and a list of items, we copy items and sort each copy in different order, and try to pack each copy and choose the best copy (i.e. the maximum sum of item area packed in the bin).

The detailed algorithm is given in Algorithm 3 and named as MixPacking. Note that in this algorithm, we need to include the implicit input of the various sort orders, these orders are used to sort the items' copies for HPA. MixPacking invokes HPA and BFA with different parameters (different order for HPA, different pattern for BFA), and returns the best result. The HPA's orders are increasing reference index (IR), decreasing reference index (DR), decreasing height decreasing width (DHDW), decreasing width decreasing height (DWDH), decreasing area (DA), decreasing perimeter (DP).

**Algorithm 3**. Mixed bin packing algorithm (MixPacking)

**Input**: a bin BIN, A list of items ITEMS
**Output**: a list of placement on bin, a list of unpacked items
MixPacking(BIN, ITEMS)
1: initialize a list of packed bins PBLIST
2: initialize an empty list UP of unpacked items
3: initialize an empty placement list PL
4: **for** order ORDER **in** HPA's orders
5: sort ITEMS with ORDER
6: PB = HPA(BIN, ITEMS)
7: add PB into PBLIST
8: **for** pattern PATTERN **in** (horizontal, vertical)
9: PB = BFA(PATTERN, BIN, ITEMS)
10: add PB into PBLIST
11: **return** the best of PBLIST

### 2.4. Backtracking algorithm for variable-sized bins packing

The MixPacking algorithm was designed to solve a single bin packing problem; for a variable-sized bin packing problem, we need to consider the arrangement of the bins. We use a backtracking algorithm to invoke the MixPacking algorithm. We name this algorithm as BTVS and its detail is given in Algorithm 5. Here, bins are selected sequentially according to the algorithm binarySearch. They are sorted in decreasing area order. The copy operation in line 3 serves to copy the current state, (the current packed bins PACKED or BINS, SET), so we can modify it during search and do not need to rollback when we backtrack. Cloned PACKED in line 9 uses sum of remained items' area and used bins' area to estimate current situation, so it is possible to achieve better result.

**Algorithm 4**. Inner backtracking procedure of BTVS

**Input**: current packed bins PACKED, a list of bins available BINS, a set of item lists with different orders named SET
**Output**: none
BT(PACKED, BINS, SET)
1: **for** each bin BIN **in** bins available
2: PB = MixPacking(BIN, SET)
3: clone PACKED
4: add PB into clone
5: **if** cloned PACKED pack all items **then**
6: **if** cloned PACKED achieves a better result **then**
7: update global result with cloned PACKED
8: **continue**
9: **if** cloned PACKED may achieve a better result **then**
10: clone BINS and SET
11: update cloned BINS and cloned SET with PB
12: BT(clone, cloned BINS, cloned SET)

For a list of bins and a set of items, BTVS incrementally builds the packed bins by invoking the MixPacking algorithm to fill items into a specified bin as much as possible, and prunes each partial solution as soon as it determines that plan cannot generate a valid solution or obtain a better solution.

The input of BTVS is an instance of the variable-sized bin packing problem, i.e. a list of bins and a list of items. The BT() function in line 5 of Algorithm 5 is the procedure stated in Algorithm 4.

**Algorithm 5**. Backtracking algorithm for variable-sized bins packing (BTVS)

**Input**: a list of bins BINS, a list of items ITEMS
**Output**: a feasible packing of the items into the bins with the aim of minimizing unutilized space in bins containing items
BTVS(BINS, ITEMS)
1: initialize a set of items list SET
2: **for** ORDER **in** ORDERS of HPA(IR, DR, DHDW, DWDH, DA, DP) and BFA(DHDW, DWDH)
3: copy ITEMS, use ORDER to sort copy, add copy into SET
4: initialize global result
5: BT(empty, BINS, SET)
6: **return** global result

## 3. Hybrid heuristic algorithm

BTVS uses a simple backtracking and greedy strategy which simply searches several items' order (DHDW, DWDH, DA, DP, etc.). In this section, we propose a method to explore the search space more thoroughly to obtain a better solution.

We focus on HPA again, since it is a powerful heuristic algorithm which allows us to do more things to obtain a better solution. More precisely, we will try to improve HPA by searching a proper order of items and a better combination of bins.

### 3.1. Simulated annealing packing (SAP)

Recall that HPA is a heuristic algorithm which generates a complete solution with a few proper orders. Here we use a simulated annealing algorithm to find a proper item order. Simulated annealing has been widely applied in packing and many other NP-hard problems, (Leung et al., 2011; Zhang, Liu, M'Hallah, & Leung, 2010). It generates a solution from a given current solution with variation, and accepts a better or worse solution with a probability which is computed from the current temperature and objective value.

**Algorithm 6**. Sequential packing (SP)

**Input**: a list of bins BINS, a list of items ITEMS
**Output**: a feasible packing of the items into the bins
SP(BINS, ITEMS)
1: initialize a list of packed bin PBLIST
2: **for** BIN **in** BINS
3: **for** $i \leftarrow 1$ **to** BIN's number **do**
4: PB = HPA(BIN, ITEMS)
5: add PB into PBLIST
6: **if** ITEMS is empty **then**
7: **return** a complete solution based on PBLIST
8: **return** an incomplete solution based on PBLIST

Sequential packing (SP) is given in Algorithm 6. The input to SP is a variable-sized bin packing problem instance. The available number of each bin types may be infinite in the original problem, but SP is designed to pack items into a set of bins with finite total area sum, so the available number of each bin types must be finite. The order of items is fixed in SP, but for different invocation of SP, the order may be different. The output of SP is a solution of the problem, and the solution may not be complete (not every items have been packed). For an incomplete solution, we use the area sum of packed items to evaluate the order of items.

Simulated annealing packing algorithm (SAP) is given in Algorithm 7. It generates a list of items with different orders and uses SP to pack items into bins one by one. Mapkob length is the number of iterations of simulated annealing search. In algorithm SP and SAP, the parameters of invoking SP and SAP include a list of bins named BINS, which is a specific bin combination chosen by algorithm binarySearch.

**Algorithm 7**. Simulated annealing packing (SAP)

**Input**: mapkob length L, a list of bins BINS, a list of items ITEMS
**Output**: a feasible packing of the items into the bins
SAP(L, BINS, ITEMS)
1: temperature $T$, $T \leftarrow T_0$
2: **while** not reach the stop criterion **do**
3: **for** $i \leftarrow 1$ **to** L **do**
4: copy ITEMS to COPY
5: randomly select two items p and q in COPY and swap them
6: current = SP(BINS, ITEMS)
7: **if** current is complete **then**
8: return current
9: **if** current > best **or** exp((current − best) / T) ⩾ rand(0, 1) **then**
10: best = current
11: ITEMS = COPY
12: **return** best

### 3.2. Hybrid heuristic algorithm

In general, there will be more than one bin combination usually which may have different performance. In addition, different instance may require different mapkob lengths to find good item orders, and up to this point, we have not discussed an effective method to determine this mapkob length.

**Algorithm 8**. Binary search packing

**Input**: mapkob length L, upper bound UB, a list of bins BINS, a list of items ITEMS
**Output**: a feasible packing of the items into the bins
binarySearch(L, UB, BINS, ITEMS)
1: LB = item area sum of ITEMS
2: Set = the set of sub bin combination of BINS whose area is in [LB, UB)
3: sort Set in decreasing area order
4: BCSet = head finite sub set of Set
5: **while** time limit not exceeded **do**
6: ind = 0
7: **while** BCSet ≠ Ø **do**
8: ind = ind + |BCSet|
9: BC = BCSet[ind]
10: remove BC from BCSet
11: sol = SAP(BC, ITEMS)
12: **if** sol is complete **then**
13: best = sol
14: remove all bin combinations whose area is large or equal to BC
15: ind = 0
16: **if** best ≠ null **then**
17: **return** best
18: BCSet = next finite sub set of Set
19: **if** BCSet = Ø **then**
20: break
21: **return** null

In the following, we propose a hybrid heuristic algorithm (HHA) to try to find a better bin combination and item order. HHA first starts with an initial length, and uses binary search similar to Zhang, Wei, Leung, and Chen (2013) as shown in Algorithm 8 to find a better bin combination that generates a complete solution with a specified order of items. If there are possible better solutions, the mapkob length is doubled, and the process is repeated until time limit is exceeded.

Binary search as shown in Algorithm 8 is designed to find a complete solution by invoking the SAP with a fixed mapkob length. The input of Algorithm 8 includes a variable-sized bin packing problem instance, a mapkob length and an upper bound of area sum of bin combination. The output is a feasible solution (better than the upper bound) or null if not found. Algorithm 8 lists all possible bin combinations and uses binary search to search a bin combination that leads a complete solution. A little difference to general binary search is when Algorithm 8 finds a complete solution, it does not return immediately, but continues to search all possible bin combinations.

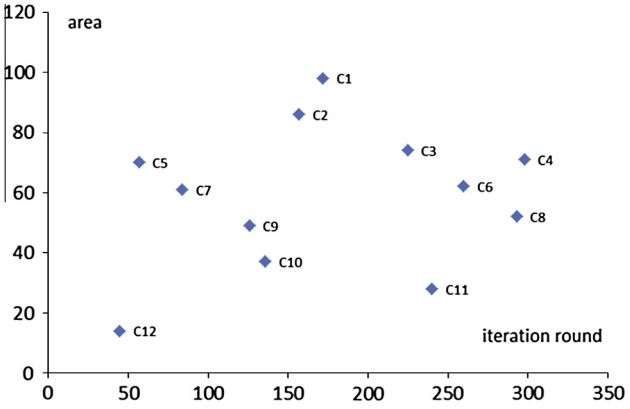


**Fig. 3.** Illustration of HHA performance.

**Algorithm 9**. Hybrid heuristic algorithm (HHA)

**Input**: a list of bins BINS, a list of items ITEMS
**Output**: a feasible packing of the items into the bins
HHA(BINS, ITEMS)
1: current = BTVS(BINS, ITEMS)
2: mapkobLength = initial mapkob length(BINS, ITEMS)
3: **while** time limit not exceeded do
4: pack = binarySearch(mapkobLength, current's area, BINS, ITEMS)
5: **if** pack is better than current **then**
6: current = pack
7: **if** current achieve the best solution **then**
8: **return** current
9: mapkobLength = mapkobLength ∗ 2
10: **return** current

The main function of HHA is given in Algorithm 9. The input of HHA is a variable-sized bin packing problem instance. The output of HHA is a complete solution. First, HHA uses BTVS to find a complete solution within a short time. Then HHA tries to find a better solution by invoking Algorithm 8 with a fixed mapkob length. If there is possibly better solution, HHA doubles the mapkob length and invokes Algorithm 8 with the doubled mapkob length. This process will repeat until time limit is exceeded or the best solution is found. In this paper, we use a fixed initial temperature, a fixed cooling rate, and use a mapkob length that is doubled after each round of HHA.

Fig. 3 reports an example of finding a good bin combination. The horizontal axis represents the iteration round, and vertical axis represents the area utilization (defined in Section 4.2) associated with a given bin combination, higher utilization means better quality. Each C label stands for a particular combination. These points describe an example of running HHA, from which we can find that HHA is an algorithm considering both time complexity and solution quality. A larger area is easier to place all items, so its computation time is smaller, but its solution quality is worse. On the contrary, a smaller area is harder to place all items, so it takes more computation time, but it may achieve a better result. For the execution of HHA in Fig. 3, at first, iteration round is set to 50. At the first pass, HHA only finds the solution with bin combination named C12. At the second pass, iteration round is doubled to 100, and the solution with C5 is found. At the third pass, we double iteration round to 200, we need to explore the C1 to C4, and find the best solution C1. In this example, HHA needs a long execution on C8, C4 and etc., if we would like to find the best solution on each bin combination from small to large one by one. Also it will spend several times of time resource on the C3 and C4, even though their area is close to the best solution C1, if we want to use the binary search with a fixed but large iteration round. We cannot assign a big iteration round at the first, or we spend too much time resource on the exploration of C11, C8, etc.

## 4. Computational experiments

These algorithms were written as sequential algorithms in Java, compiled and run using oracle JDK7. Experiments were performed

on an Intel Core i5-2400 CPU (quad core, 3.10 gigahertz) and 8 gigabytes RAM running the Centos 6 Linux operating system.

For this experiment, algorithms are run in a time limit, BTVS's limit is 10 CPU seconds or it searches all possible solutions, HHA's limit is 60 CPU seconds or it finds the best solution. The initial mapkob length of HHA uses the number of items.

### 4.1. Benchmarks

We use three set of benchmark data to evaluate the effectiveness of HHA. First benchmark data is for 2DVSBPP, we compare HHA with various algorithms in Ortmann et al. (2010). And we compare HHA with different algorithms with correspond benchmarks, these two benchmarks are for 2DBPP.

We use a large set of benchmark data to evaluate the effectiveness of HHA to the two-stage algorithm with different strip packing algorithms in Ortmann et al. (2010). There are several online repositories of benchmark data for packing problems, but benchmarks for variable-sized bin packing problem are fewer than the popular strip packing problem. There are only data set M from Hopper and Turton (2002) and data set PS from Pisinger and Sigurd (2005) for this problem.

From the benchmark instances for 2DBPP, Pisinger and Sigurd (2005) generated new 2DVSBPP instances with variable bin cost. Each original instance contains a type of bin without limit of number, and the corresponding instance adds five new 5 types of bin without limit of number, the new added bin's dimensions are selected from range $[W/2,W] \times [H/2,H]$ independently, $W$ and $H$ is the original bin type's dimension. Ortmann et al. (2010) remove the bin costs in their paper. They created benchmarks for testing the effects on nice and pathological data, and their benchmarks are generated in the same manner of Wang and Valenzela (2001) and are named as Nice and Path. Since Nice and Path's generation algorithm begins with a square rectangle of dimension $1000 \times 1000$, each case should have an optimal solution. Both include combinations of 2–6 bin types and 25–500 items (these is no combination of 6 bin types and 25 items), each combination contains 5 problems.

For 2DBPP with guillotine cuts, the bins are of same size, one data set (BWMV) include classes 1–6 proposed by Berkey and Wang (1987) and classes 7–10 proposed by Martello and Vigo (1998). Each class has 10 instances of size 20, 40, 60, 80, and 100 items. This data set has a total of 500 instances. Classes differ in the bin size ranging from$10 \times 10$ to $300 \times 300$. Item dimensions are random integers from uniform distributions in various ranges. Another data set (Ngcutfsd) includes classes $d = 1, 2, 3$ proposed by Beasley (2004). For each class, seven problem sizes are considered: $n = 40, 50, 100, 150, 250, 500$ and 1000. For each problem type, 10 instances are included. The bin sizes are $100 \times 100$. Two data sets are well-known and classic, and then they are widely used to test the performance of different algorithms for 2DBPP.

### 4.2. Numerical results 2DVSBPP

In order to test the effectiveness of HHA, experiments use utilization to evaluate it and other algorithms. The utilization μ of a solution is the total area of items divided by the area of the bins used, that is

$$\mu = \frac{\text{total area of items}}{\text{area of the bins used}} \times 100\%.$$

With area of bins used decreases, wasted space decreases and the utilization increases. If a solution's utilization is 90%, that means 10% space wasted. Thus the goal is to maximize the utilization.

**Table 2**
Summary of the average packing utilizations achieved for 2DVSBPP.

| | Source | M1 | M2 | M3 | PS | Nice | Path | Average utilization |
|---|---|---|---|---|---|---|---|---|
| $n$ | | 100 | 100 | 150 | 60 | 231 | 231 | |
| FFDH | Coffman et al. (1980) | 93.5 | 87.5 | 92 | 80.7 | 81.8 | 80.9 | 86.0667 |
| BFDH | Coffman & Shor, 1990 | 93.5 | 88.8 | 92.6 | 81 | 81.7 | 81.2 | 86.4667 |
| JOIN | Martello et al. (2003) | 86.8 | 82.8 | 85.5 | 78 | 78.6 | 78.7 | 81.7333 |
| FCOG | Lodi et al. (1999a) Lodi et al. (1999b) | 94.9 | 89.6 | 93.6 | 81.9 | 82.7 | 85.3 | 88 |
| BFDH* | Bortfeldt (2006) | 94.9 | 89.6 | 93.6 | 81.7 | 82.5 | 83.5 | 87.6333 |
| SAS | Ntene (2007) Ntene and van Vuuren (2009) | 83.5 | 81.7 | 86.8 | 74.9 | 76.8 | 76.9 | 80.1 |
| SASm | Ortmann et al. (2010) | 91.6 | 88 | 91.8 | 79 | 78.4 | 78.5 | 84.55 |
| BFS | Ortmann et al. (2010) | 95.5 | 90 | 94.9 | 82 | 82.6 | 85 | 88.3333 |
| HHA | | 98.4 | 95.6 | 97.4 | 87.8 | 91.5 | 94.6 | 94.2167 |

**Table 3**
Average packing utilizations achieved for benchmark in Pisinger and Sigurd (2005).

| Class | $n$ | FFDH | BFDH | JOIN | FCOG | BFDH* | SAS | SASm | BFS | HHA |
|---|---|---|---|---|---|---|---|---|---|---|
| I | 60 | 86.3 | 86.6 | 83 | 87.3 | 87.4 | 79.4 | 86.3 | 88.6 | 91.5 |
| II | 60 | 83.7 | 83.7 | 80.9 | 85.2 | 85 | 80.1 | 82.2 | 85.1 | 96.3 |
| III | 60 | 81.1 | 81.7 | 76.7 | 81.9 | 81.9 | 69.6 | 75.7 | 82.2 | 86.6 |
| IV | 60 | 80 | 80 | 79.1 | 82.7 | 82.1 | 76 | 78 | 82 | 91.7 |
| V | 60 | 80.4 | 81.1 | 77.6 | 81.3 | 81.2 | 72.6 | 78.3 | 81.4 | 84.6 |
| VI | 60 | 79.1 | 79.3 | 78.3 | 80.7 | 80.5 | 76.1 | 77.1 | 79.5 | 90.2 |
| VII | 60 | 79.9 | 80.2 | 79.6 | 80.8 | 80.6 | 74 | 80.4 | 80.9 | 86.9 |
| VIII | 60 | 80.7 | 81.1 | 74.2 | 81.3 | 81.2 | 76.4 | 79.5 | 81.6 | 85.9 |
| IX | 60 | 72.8 | 72.6 | 72.1 | 72.6 | 72.8 | 71.6 | 72.9 | 73 | 74.3 |
| X | 60 | 83.3 | 83.8 | 79.3 | 85.4 | 84.6 | 73.2 | 79.4 | 85.5 | 90.3 |
| 20 | 20 | 73.5 | 73.7 | 71.7 | 75.4 | 75.4 | 70 | 72.3 | 75.2 | 82.9 |
| 40 | 40 | 79.1 | 79.3 | 76.9 | 80.3 | 79.8 | 74.6 | 78 | 80.1 | 88.5 |
| 60 | 60 | 81.7 | 82 | 79.2 | 83 | 82.7 | 74.9 | 80 | 83.1 | 89.3 |
| 80 | 80 | 84 | 84.1 | 80.9 | 84.6 | 84.4 | 76.7 | 81.6 | 85 | 88.8 |
| 100 | 100 | 85.3 | 85.8 | 81.9 | 86.3 | 86.3 | 78.3 | 83.1 | 86.5 | 89.7 |
| Average utilization | | 80.7267 | 81.0000 | 78.0933 | 81.9200 | 81.7267 | 74.9000 | 78.9867 | 81.9800 | **87.8333** |

**Table 4**
Average utilizations achieved for benchmark in Ortmann et al. (2010).

| Class | $n$ | FFDH | BFDH | JOIN | FCOG | BFDH* | SAS | SASm | BFS | HHA |
|---|---|---|---|---|---|---|---|---|---|---|
| Nice25i | 25 | 73.9 | 73.6 | 70.6 | 73.9 | 73.6 | 68.3 | 71.8 | 73.6 | 94.3 |
| Nice50i | 50 | 76.7 | 76.7 | 73.3 | 77.9 | 77.7 | 70.8 | 73.1 | 77.8 | 89.2 |
| Nice100i | 100 | 79.4 | 79.4 | 77.5 | 79.9 | 79.4 | 75.7 | 76.3 | 79.4 | 88.3 |
| Nice200i | 200 | 82 | 82 | 81.5 | 84.5 | 84.5 | 78.5 | 79.4 | 84.5 | 91 |
| Nice300i | 300 | 85.8 | 85.8 | 83.3 | 86 | 86.5 | 80.2 | 81.7 | 86.8 | 92 |
| Nice400i | 400 | 85.1 | 85.1 | 82.7 | 86.6 | 85.7 | 80.2 | 81 | 85.7 | 92.8 |
| Nice500i | 500 | 87.2 | 87.2 | 84.8 | 87.7 | 87.7 | 81.8 | 83.8 | 87.7 | 93.1 |
| Path25i | 25 | 76.3 | 76.3 | 73.9 | 77.9 | 77.6 | 72.2 | 74.4 | 78.3 | 97.2 |
| Path50i | 50 | 76.4 | 78.6 | 74.2 | 81.6 | 79.4 | 72.4 | 75.6 | 81.9 | 94.9 |
| Path100i | 100 | 79.7 | 79.7 | 77.8 | 83.2 | 81.3 | 72.4 | 75.8 | 83.5 | 93.5 |
| Path200i | 200 | 84.1 | 84 | 81.8 | 88 | 85.9 | 77.7 | 79 | 87.5 | 93.3 |
| Path300i | 300 | 82.9 | 82.9 | 82.7 | 87 | 86 | 81.4 | 81.7 | 87.3 | 94.9 |
| Path400i | 400 | 82.7 | 82.7 | 82.3 | 89.6 | 87 | 79.9 | 80.1 | 88.5 | 94.8 |
| Path500i | 500 | 82.9 | 82.9 | 81.2 | 88.7 | 86.4 | 81.3 | 82.4 | 86.7 | 94 |
| Nice2bin | 225 | 75 | 75 | 73.4 | 75.8 | 75.8 | 71.3 | 72 | 75.8 | 83.2 |
| Nice3bin | 225 | 81.5 | 81.5 | 79.1 | 82.8 | 82.8 | 76.3 | 77.7 | 82.9 | 92 |
| Nice4bin | 225 | 83.2 | 83 | 80.9 | 83.8 | 83.4 | 78.4 | 81.3 | 83.6 | 94.1 |
| Nice5bin | 225 | 83.8 | 83.8 | 80.7 | 85 | 84.4 | 77.7 | 79.1 | 84.6 | 94.1 |
| Nice6bin | 258 | 85.3 | 85.3 | 83.2 | 86.2 | 86.3 | 80.5 | 82.1 | 86 | 94.3 |
| Path2bin | 225 | 69.7 | 70.2 | 68.5 | 74.7 | 71.6 | 70.8 | 70.6 | 74 | 87.4 |
| Path3bin | 225 | 80.1 | 80.1 | 78.9 | 86 | 83.9 | 75.2 | 76.3 | 85.2 | 94.9 |
| Path4bin | 225 | 82.9 | 83.3 | 81.3 | 86.6 | 85.5 | 74.7 | 78.3 | 86.3 | 96.5 |
| Path5bin | 225 | 84.3 | 84.8 | 83.2 | 89.5 | 88 | 79.7 | 81.9 | 89.8 | 96.9 |
| Path6bin | 258 | 88.3 | 88.4 | 85.4 | 90.5 | 89.5 | 85.2 | 86.9 | 90.5 | 97.6 |
| Average utilization | | 81.2167 | 81.3458 | 79.2583 | 83.8917 | 82.9125 | 76.7750 | 78.4292 | 83.6625 | **93.0958** |

**Table 5**
The performance ratio of GBL, A-B and HHA for data set Ngcutfsd.

| Instance | $n$ | Class $d = 1$ | | | Class $d = 2$ | | | Class $d = 3$ | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | $R_{GBL}$ | $R_{A\text{-}B}$ | $R_{HHA}$ | $R_{GBL}$ | $R_{A\text{-}B}$ | $R_{HHA}$ | $R_{GBL}$ | $R_{A\text{-}B}$ | $R_{HHA}$ |
| Ngcutfsd_11 | 40 | 1.200 | 1.200 | 1.200 | 1.200 | 1.200 | 1.200 | 1.143 | 1.000 | 1.000 |
| Ngcutfsd_12 | 40 | 1.222 | 1.111 | 1.111 | 1.333 | 1.222 | 1.222 | 1.333 | 1.167 | 1.167 |
| Ngcutfsd_13 | 40 | 1.222 | 1.111 | 1.111 | 1.182 | 1.091 | 1.091 | 1.286 | 1.000 | 1.000 |
| Ngcutfsd_41 | 50 | 1.154 | 1.154 | 1.154 | 1.333 | 1.111 | 1.111 | 1.200 | 1.200 | 1.100 |
| Ngcutfsd_42 | 50 | 1.167 | 1.083 | 1.083 | 1.222 | 1.222 | 1.111 | 1.125 | 1.125 | 1.000 |
| Ngcutfsd_43 | 50 | 1.167 | 1.167 | 1.083 | 1.214 | 1.214 | 1.143 | 1.143 | 1.143 | 1.143 |
| Ngcutfsd_71 | 100 | 1.174 | 1.130 | 1.130 | 1.182 | 1.136 | 1.136 | 1.133 | 1.067 | 1.067 |
| Ngcutfsd_72 | 100 | 1.115 | 1.115 | 1.077 | 1.095 | 1.048 | 1.048 | 1.111 | 1.111 | 1.056 |
| Ngcutfsd_73 | 100 | 1.125 | 1.125 | 1.083 | 1.095 | 1.095 | 1.048 | 1.133 | 1.067 | 1.067 |
| Ngcutfsd_101 | 150 | 1.111 | 1.056 | 1.056 | 1.097 | 1.065 | 1.065 | 1.095 | 1.048 | 1.048 |
| Ngcutfsd_102 | 150 | 1.121 | 1.091 | 1.061 | 1.100 | 1.067 | 1.033 | 1.150 | 1.100 | 1.100 |
| Ngcutfsd_103 | 150 | 1.097 | 1.065 | 1.065 | 1.094 | 1.094 | 1.063 | 1.154 | 1.077 | 1.077 |
| Ngcutfsd_131 | 250 | 1.086 | 1.069 | 1.069 | 1.132 | 1.094 | 1.094 | 1.077 | 1.051 | 1.051 |
| Ngcutfsd_132 | 250 | 1.143 | 1.095 | 1.095 | 1.077 | 1.058 | 1.058 | 1.103 | 1.051 | 1.051 |
| Ngcutfsd_133 | 250 | 1.098 | 1.082 | 1.066 | 1.095 | 1.048 | 1.048 | 1.079 | 1.053 | 1.053 |
| Ngcutfsd_161 | 500 | 1.091 | 1.066 | 1.066 | 1.051 | 1.040 | 1.040 | 1.052 | 1.039 | 1.039 |
| Ngcutfsd_162 | 500 | 1.083 | 1.058 | 1.074 | 1.065 | 1.043 | 1.065 | 1.051 | 1.038 | 1.051 |
| Ngcutfsd_163 | 500 | 1.080 | 1.072 | 1.096 | 1.067 | 1.034 | 1.056 | 1.057 | 1.043 | 1.043 |
| Ngcutfsd_191 | 1000 | 1.079 | 1.041 | 1.071 | 1.054 | 1.034 | 1.044 | 1.039 | 1.026 | 1.033 |
| Ngcutfsd_192 | 1000 | 1.070 | 1.035 | 1.065 | 1.044 | 1.027 | 1.038 | 1.034 | 1.027 | 1.027 |
| Ngcutfsd_193 | 1000 | 1.081 | 1.051 | 1.064 | 1.056 | 1.036 | 1.056 | 1.034 | 1.027 | 1.034 |
| Average ratio | | 1.127 9 | 1.094 1 | **1.089 5** | 1.132 8 | 1.094 2 | **1.084 3** | 1.120 6 | 1.069 5 | **1.057 5** |

Table 2 gives a summary of the average packing utilizations achieved by different algorithms for 2DVSBPP. Where the name of algorithms is given in the first row, the second row is source on algorithms. Table 3 gives average packing utilizations achieved for VSBP on benchmark in Pisinger and Sigurd (2005). And Table 4 reports average packing utilizations achieved for VSBP on benchmark in Ortmann et al. (2010). Average denotes the average utilization of the considered problems. All of these strip algorithms observes the guillotine constraint. Except for HHA, results of other algorithms are cited from Ortmann et al. (2010).

Comparing the results (see *Average utilization*) in Tables 2, 3, and 4, we can observe that HHA achieves a better result than other algorithms, especially for the data sets Nice and Path. The improvement on average utilization is huge. For data sets M (M1-3) and PS, HHA achieves a better result, but its improvement on these two data sets is not good enough comparing to Nice and Path. The reasons are that M and PS are not designed with an optimal solution of full utilization of bins, their solutions are irregular, so it is difficult to find an optimal solution.

### 4.3. Numerical results 2DBPP of same bin size

A-B (Polyakovsky & M'Hallah, 2009) and CHBP (Charalambous & Fleszar, 2011) are the best methods available for 2DBPP with guillotine cuts in the literature. The performance of A-B, CHBP

**Table 6**
The performance ratio of TS, A-B,CHBP and HHA for data set BWMV.

| BWMV | $n$ | $R_{TS}$ | $R_{A\text{-}B}$ | $R_{CHBP}$ | $R_{HHA}$ |
|---|---|---|---|---|---|
| Class 1 | 20 | 1.11 | 1.01 | 1.00 | 1.00 |
| | 40 | 1.08 | 1.02 | 1.00 | 1.00 |
| | 60 | 1.05 | 1.03 | 1.02 | 1.02 |
| | 80 | 1.04 | 1.00 | 1.00 | 1.00 |
| | 100 | 1.05 | 1.02 | 1.01 | 1.01 |
| Average ratio | | 1.066 | 1.017 | 1.008 | 1.006 |
| Class 2 | 20 | 1.00 | 1.00 | 1.00 | 1.00 |
| | 40 | 1.10 | 1.10 | 1.10 | 1.00 |
| | 60 | 1.15 | 1.00 | 1.05 | 1.00 |
| | 80 | 1.07 | 1.00 | 1.07 | 1.00 |
| | 100 | 1.03 | 1.00 | 1.00 | 1.00 |
| Average ratio | | 1.070 | 1.020 | 1.043 | 1.000 |
| Class 3 | 20 | 1.18 | 1.05 | 1.03 | 1.03 |
| | 40 | 1.12 | 1.08 | 1.07 | 1.03 |
| | 60 | 1.07 | 1.07 | 1.03 | 1.03 |
| | 80 | 1.08 | 1.06 | 1.05 | 1.02 |
| | 100 | 1.09 | 1.05 | 1.04 | 1.03 |
| Average ratio | | 1.108 | 1.062 | 1.046 | 1.029 |
| Class 4 | 20 | 1.00 | 1.00 | 1.00 | 1.00 |
| | 40 | 1.10 | 1.00 | 1.00 | 1.00 |
| | 60 | 1.20 | 1.10 | 1.10 | 1.10 |
| | 80 | 1.10 | 1.10 | 1.10 | 1.07 |
| | 100 | 1.10 | 1.03 | 1.07 | 1.03 |
| Average ratio | | 1.100 | 1.047 | 1.053 | 1.040 |
| Class 5 | 20 | 1.13 | 1.02 | 1.00 | 1.00 |
| | 40 | 1.09 | 1.04 | 1.02 | 1.00 |
| | 60 | 1.07 | 1.04 | 1.02 | 1.01 |
| | 80 | 1.08 | 1.06 | 1.03 | 1.03 |
| | 100 | 1.09 | 1.06 | 1.04 | 1.03 |
| Average ratio | | 1.092 | 1.043 | 1.021 | 1.014 |
| Class 6 | 20 | 1.00 | 1.00 | 1.00 | 1.00 |
| | 40 | 1.50 | 1.40 | 1.40 | 1.30 |
| | 60 | 1.10 | 1.05 | 1.05 | 1.05 |
| | 80 | 1.00 | 1.00 | 1.00 | 1.00 |
| | 100 | 1.10 | 1.10 | 1.07 | 1.10 |
| Average ratio | | 1.140 | 1.110 | 1.103 | 1.090 |
| Class 7 | 20 | 1.08 | 1.04 | 1.00 | 1.00 |
| | 40 | 1.07 | 1.05 | 1.03 | 1.02 |
| | 60 | 1.05 | 1.04 | 1.02 | 1.02 |
| | 80 | 1.05 | 1.05 | 1.04 | 1.04 |
| | 100 | 1.04 | 1.03 | 1.02 | 1.02 |
| Average ratio | | 1.058 | 1.043 | 1.023 | 1.018 |
| Class 8 | 20 | 1.12 | 1.02 | 1.00 | 1.00 |
| | 40 | 1.04 | 1.03 | 1.02 | 1.01 |
| | 60 | 1.03 | 1.04 | 1.03 | 1.02 |
| | 80 | 1.03 | 1.03 | 1.01 | 1.01 |
| | 100 | 1.04 | 1.04 | 1.02 | 1.02 |
| Average ratio | | 1.052 | 1.032 | 1.015 | 1.011 |
| Class 9 | 20 | 1.00 | 1.00 | 1.00 | 1.00 |
| | 40 | 1.01 | 1.00 | 1.00 | 1.00 |
| | 60 | 1.01 | 1.00 | 1.00 | 1.00 |
| | 80 | 1.01 | 1.00 | 1.00 | 1.00 |
| | 100 | 1.01 | 1.00 | 1.00 | 1.00 |
| Average ratio | | 1.008 | 1.000 | 1.001 | 1.000 |
| Class 10 | 20 | 1.14 | 1.05 | 1.05 | 1.02 |
| | 40 | 1.09 | 1.03 | 1.00 | 1.00 |
| | 60 | 1.08 | 1.08 | 1.05 | 1.05 |
| | 80 | 1.10 | 1.06 | 1.06 | 1.06 |
| | 100 | 1.07 | 1.08 | 1.07 | 1.05 |
| Average ratio | | 1.096 | 1.058 | 1.044 | 1.033 |
| Total average ratio | | 1.079 | 1.043 | 1.036 | **1.024** |

and HHA is verified by using the data sets BWMV and Ngcutfsd. The performance ratios $R_A$ on algorithm A are computed as follows: $R_A = Z_A/LB$, where $Z_A$ is the average number of bins obtained by algorithm A, $LB$ is the lower bound for the minimum number of bins. The running time of HHA does not exceed 60 seconds.

**Table 7**
The performance for data set BWMV.

| | Avg. dev. | Avg. time (seconds) | Processor |
|---|---|---|---|
| TS | 1.079 | 32.55 | Silicon graphics |
| A-B | 1.043 | | Athlon XP 2800 |
| CHBP | 1.036 | 0.033 | Intel i3 2.13 GHz |
| HHA | 1.024 | 38.582 | Intel Core i5-2400 |

Table 5 shows the results of GBL, A-B and HHA for 2DBPP with orientation and guillotine cuts constraints. Column 1 refers to the instance where $d = 1, 2, 3$ indicates the class of the instance whereas column 2 refers to the problem size $n$. The performance ratios of GBL and A-B are taken from Polyakovsky and M'Hallah (2009), where A-B's solution is the global optimum of 50 independent runs and the running time is not reported.

From Table 5, we can find that both A-B and HHA achieve the better results on some instances. A-B achieves better results on some instances with large number of items while HHA achieves better results on instances with small number of items. For Class $d = 1$, Class $d = 2$ and Class $d = 3$, the average performance ratio (*Average ratio*) of HHA is 1.0895, 1.0843 and 1.0575 (see bold number in Table 5) respectively. Although HHA is not specially developed to solve two-dimensional bin packing of the same size, HHA still outperforms GBL and A-B on average.

Table 6 shows the results of TS, A-B, CHBP and HHA for two-dimensional bin packing of the same size with orientation and guillotine cuts constraints. The performance ratios of TS, A-B and CHBP are taken from Charalambous and Fleszar (2011). From Table 6, we can find that HHA achieves the best results for each class problem, therefore, HHA outperforms TS, A-B and CHBP on average.

Algorithms in Tables 2, 3, 4 are simple heuristic algorithm, the computing time of these algorithms is much less than HHA. As algorithms GBL and A-B in Polyakovsky and M'Hallah (2009), their computing time is small, and they did not provide a table about it, so we did not compare about it. For algorithm CHBP for data set BWMV, it provides an average time of it, so we add a new Table 7 about this data set. The quality of solutions is reported in terms of average deviations from lower bounds (Avg. dev.) and the total number of bins used. Deviation for each instance is computed as the number of bins divided by the lower bound on the number of bins. Average processing times (Avg. time) is reported in seconds. HHA can find better solutions than other algorithms within a reasonable time.

## 5. Conclusions

This paper tackles 2DVSBPP|O|G. The importance of this problem lies in its widespread applications in transport, loading and industry. Unfortunately, this problem belongs to the class of NP-hard problems. To solve this problem, a hybrid heuristic algorithm is proposed in this paper. The main contributions of this paper propose a hybrid heuristic algorithm based on simulated annealing and binary search for 2DVSBPP|O|G. To the best of our knowledge, it is the first hybrid meta-heuristic algorithm for 2DVSBPP|O|G. The second main contribution is the use of backtracking and an improved score strategy. The third contribution is the improvement of the scoring strategy. Computational results are very encouraging and clearly show HHA outperforms the exist algorithms. As perspective, we want to extend HHA to solve three-dimensional variable-sized bin packing problems or other variants.

## 6. Abbreviation

| Abbr. | Name | Source |
|---|---|---|
| 2DBPP | Two-dimensional bin packing problem | |
| 2DVSBPP | Two-dimensional variable-sized bin packing problem | |
| SAS | Size-alternating stack algorithm | Bortfeldt (2006) |
| BFDH | Best-fit decreasing height algorithm | Coffman and Shor (1990) |
| FFDH | First-fit decreasing height algorithm | Coffman et al. (1980) |
| FC | Floor-ceiling algorithm | Lodi et al. (1999a) |
| BFDH* | Improved best-fit decreasing height algorithm | Lodi et al. (1999b) |
| HFC | Hybrid floor-ceiling algorithm | Lodi et al. (2002) |
| JOIN | | Martello et al. (2003) |
| SASm | Modified size-alternating stack algorithm | Ntene and van Vuuren (2009) |
| BFS | Best-fit with stacking algorithm | Ortmann et al. (2010) |
| BFA | Best Fit algorithm | This article |
| binarySearch | Binary search packing | |
| HHA | Hybrid heuristic algorithm | |
| HPA | Heuristic packing algorithm | |
| MixPacking | Mixed bin packing algorithm | |
| SAP | Simulated annealing packing | |
| SP | Sequential packing | |
| BTVS | Backtracking algorithm for variable-sized bins packing | |
| DA | Decreasing area | Sort order |
| DHDW | Decreasing height decreasing width | |
| DP | Decreasing perimeter | |
| DR | Decreasing reference index | |
| DWDH | Decreasing width decreasing height | |
| IR | Increasing reference index | |

## Acknowledgments

This work has been supported by the National Nature Science Foundation of China (jointly by Grant No. 61272003) and the Project Sponsored by the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry.

## References

Alvarez-Valdes, R., Parreño, F., & Tamarit, J. M. (2013). A GRASP/Path relinking algorithm for two- and three-dimensional multiple bin-size bin packing problems. *Computers & Operations Research, 40*(12), 3081–3090. http://dx.doi.org/10.1016/j.cor.2012.03.016.

Beasley, J. E. (2004). A population heuristic for constrained two-dimensional non-guillotine cutting. *European Journal of Operational Research, 156*(3), 601–627. http://dx.doi.org/10.1016/S0377-2217(03)00139-5.

Berkey, J., & Wang, P. (1987). Two-dimensional finite bin-packing algorithms. *Journal of the Operational Research Society*, 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.

Charalambous, C., & Fleszar, K. (2011). A constructive bin-oriented heuristic for the two-dimensional bin packing problem with guillotine cuts. *Computers & Operations Research, 38*(10), 1443–1451. http://dx.doi.org/10.1016/j.cor.2010.12.013.

Coffman, J., Edward, G., Garey, M. R., Johnson, D. S., & Tarjan, R. E. (1980). Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing, 9*(4), 808–826.

Coffman, E., Jr., & Shor, P. (1990). Average-case analysis of cutting and packing in two dimensions. *European Journal of Operational Research, 44*(2), 134–144.

Fleszar, K. (2013). Three insertion heuristics and a justification improvement heuristic for two-dimensional bin packing with guillotine cuts. *Computers & Operations Research, 40*(1), 463–474. http://dx.doi.org/10.1016/j.cor.2012.07.016.

Hopper, E., & Turton, B. (2002). Problem generators for rectangular packing problems. *Studia Informatica Universalis, 2*(1), 123–136.

Kang, J., & Park, S. (2003). Algorithms for the variable sized bin packing problem. *European Journal of Operational Research, 147*(2), 365–372.

Leung, S. C., Zhang, D., & Sim, K. M. (2011). A two-stage intelligent search algorithm for the two-dimensional strip packing problem. *European Journal of Operational Research, 215*(1), 57–69.

Liu, Y., Chu, C., & Wang, K. (2011). A dynamic programming-based heuristic for the variable sized two-dimensional bin packing problem. *International Journal of Production Research, 49*(13), 3815–3831. http://dx.doi.org/10.1080/00207543.2010.501549.

Lodi, A., Martello, S., & Vigo, D. (1999a). Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing, 11*(4), 345–357.

Lodi, A., Martello, S., & Vigo, D. (1999b). Neighborhood search algorithm for the guillotine non-oriented two-dimensional bin packing problem. In *Meta-Heuristics* (pp. 125–139). Springer.

Lodi, A., Martello, S., & Vigo, D. (2002). Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics, 123*(1–3), 379–396. http://dx.doi.org/10.1016/S0166-218X(01)00347-X.

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

Martello, S., & Vigo, D. (1998). Exact solution of the two-dimensional finite bin packing problem. *Management Science, 44*(3), 388–399. http://dx.doi.org/10.1287/mnsc.44.3.388.

Ntene, N. (2007). *An algorithmic approach to the 2D oriented strip packing problem*. University of Stellenbosch.

Ntene, N., & van Vuuren, J. H. (2009). A survey and comparison of guillotine heuristics for the 2D oriented offline strip packing problem. *Discrete Optimization, 6*(2), 174–188.

Omar, M. K., & Ramakrishnan, K. (2013). Solving non-oriented two dimensional bin packing problem using evolutionary particle swarm optimisation. *International Journal of Production Research, 51*(20), 6002–6016. http://dx.doi.org/10.1080/00207543.2013.791754.

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.

Pisinger, D., & Sigurd, M. (2005). The two-dimensional bin packing problem with variable bin sizes and costs. *Discrete Optimization, 2*(2), 154–167.

Pisinger, D., & Sigurd, M. (2007). Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem. *INFORMS Journal on Computing, 19*(1), 36–51. http://dx.doi.org/10.1287/ijoc.1060.0181.

Polyakovsky, S., & M'Hallah, R. (2009). An agent-based approach to the two-dimensional guillotine bin packing problem. *European Journal of Operational Research, 192*(3), 767–781. http://dx.doi.org/10.1016/j.ejor.2007.10.020.

Valério de Carvalho, J. M. (2002). LP models for bin packing and cutting stock problems. *European Journal of Operational Research, 141*(2), 253–273. http://dx.doi.org/10.1016/S0377-2217(02)00124-8.

Wang, P. Y., & Valenzela, C. L. (2001). Data set generation for rectangular placement problems. *European Journal of Operational Research, 134*(2), 378–391.

Wei, L., Oon, W.-C., Zhu, W., & Lim, A. (2013). A goal-driven approach to the 2D bin packing and variable-sized bin packing problems. *European Journal of Operational Research, 224*(1), 110–121.

Zhang, D., Liu, Y., M'Hallah, R., & Leung, S. C. (2010). A simulated annealing with a new neighborhood structure based algorithm for high school timetabling problems. *European Journal of Operational Research, 203*(3), 550–558.

Zhang, D., Wei, L., Leung, S. C., & Chen, Q. (2013). A binary search heuristic algorithm based on randomized local search for the rectangular strip-packing problem. *INFORMS Journal on Computing, 25*(2), 332–345.