Computers and Operations Research 80 (2017) 113–127

Contents lists available at ScienceDirect

Computers and Operations Research

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

An improved skyline based heuristic for the 2D strip packing problem and its efficient implementation

Lijun Wei[a,b,*], Qian Hu[c,d], Stephen C.H. Leung[e], Ning Zhang[f,g]

[a] School of Information Technology, Jiangxi University of Finance and Economics, Nanchang, Jiangxi 330013, China
[b] City University of Hong Kong Shenzhen Research Institute (CityUSRI), Shenzhen 518057, China
[c] International Center of Management Science and Engineering, School of Management and Engineering, Nanjing University, Nanjing 210093, China
[d] Department of Industrial & Systems Engineering, National University of Singapore, 117576, Singapore
[e] Faulty of Engineering, The University of Hong Kong, Pokfulam Road, 999077, Hong Kong
[f] Department of Economics, Jinan University, Guangzhou, Guangdong 510632, China
[g] Institute of Resource, Environment and Sustainable Development, Jinan University, Guangzhou, Guangdong 510632, China

ARTICLE INFO

Article history:
Received 8 February 2016
Revised 22 November 2016
Accepted 24 November 2016
Available online 29 November 2016

Keywords:
Cutting
Packing
Heuristics
Random local search
Best-fit

ABSTRACT

The best-fit heuristic by Burke et al. (2004) is a simple but effective approach for the 2D Strip Packing (2DSP) problem. In this paper, we propose an improved best-fit heuristic for the 2DSP. Instead of selecting the rectangle with the largest width, we use the fitness number to select the best rectangle fitting into the gap. An efficient implementation pattern with a time complexity of $O(n \log n)$ (n is the number of rectangles) is provided for the improved best-fit heuristic. A simple random local search is used to improve the results by trying different sequences. The experiment on the benchmark test sets shows that the final approach is both effective and efficient.

1. Introduction

The 2D Strip Packing (2DSP) problem is a fundamental problem in cutting and packing literature. Given a set of n rectangles with dimensions $w_i \times h_i$, $i = 1, \ldots, n$, the task is to orthogonally pack the rectangles without overlap into a large rectangular strip with width W, in order to minimize the height of the packing. All dimensions are assumed integral. This paper considers fixed orientation variant of the 2DSP, in which the rotation of the rectangle is not allowed.

Under the improved typology of cutting and packing problems by Wascher et al. [2], the 2DSP is a two-dimensional open dimension problem (2D-ODP). The 2DSP is NP-hard in the strong sense. The exact algorithms [3–8] can only solve the small size instances. Thus, most of the research focus on the heuristic to provide a good solution in a reasonable time.

Various constructive heuristic algorithms based on different methodologies have been presented for solving the 2DSP. The renowned method is the Bottom-Left (BL) method proposed by Baker et al. [9]. In the BL, each rectangle is placed sequentially by pushing it downwards, then leftwards as much as possible. The Bottom-Left Fill (BLF) method by Chazelle [10] generalizes the BL concept. In the BLF, each rectangle is placed at the bottom-left position. Hopper and Turton [11] proposed an approach called the Bottom-Left-Decreasing (BLD) that executes the BL heuristic on a number of different sequences and selects the best result. Zang et al. [12]. used a Recursive Heuristic (RH) based on a divide-and-conquer principle. Cui et al. [13] suggested a recursive branch-and-bound algorithm for the 2DSP with the guillotine constraint. He et al. [14] proposed a deterministic heuristic algorithm for rotatable 2DSP based on the action space. Wang and Chen [15] described a heuristic algorithm based on the residual space.

Both the BL and BLF select the rectangle first (the first unpacked one) and then find its best position, whilst another common approach selects position first and then finds the best-fit rectangle. Burke et al. [1] suggested one such approach that firstly selects the bottom-left "gap" and then finds the Best Fit (BF) rectangle via a priority rule. Aşik and Özcan [16] extended the BF by introducing a novel bidirectional best-fit heuristic. More recently, Verstichel et al. [17] proposed an improved BF heuristic, which improves the solution quality on all test sets compared to the original BF. Leung et al. [18] enhanced the BF by introducing a novel scoring rule.

Another approach is to select the placement (i.e., a pair of position and rectangle) based on some measures. Wei et al. [19]

* Corresponding author at: School of Information Technology, Jiangxi University of Finance and Economics, Nanchang, Jiangxi 330013, China.
E-mail addresses: villagerwei@gmail.com (L. Wei), huqian@nju.edu.cn (Q. Hu).

http://dx.doi.org/10.1016/j.cor.2016.11.024
0305-0548/

presented a Least-Waste-First (LWF) strategy to select the best placement. Wu et al. [20] introduced another measure called Less Flexibility First (LFF). Wei et al. [21] proposed a skyline based heuristic and suggested a set of priority rules.

Based on the constructive heuristic, the meta-heuristics are introduced to further improve the solution. Liu and Teng [22] presented a genetic algorithm to improve the solution of the BL. Hopper and Turton [11] compared the performances of combing BL or BLF with different meta-heuristics (simulated annealing, tabu search and genetic algorithm). Zang et al. [23] studied meta-heuristic (simulated annealing and genetic algorithm) enhancements of the RH heuristic. Burke et al. [24] introduced a simulated annealing enhancement of the BF, which optimizes the placement in the final stage of packing. Leung et al. [18] presented a two-stage Intelligent Search Algorithm (ISA) for their improved BF. Thereafter, Yang et al. [25] proposed a Simple Randomized Algorithm (SRA) based on the ISA. Wei et al. [26] proposed an efficient Intelligent search Algorithm (IA) by introducing an improved scoring rule to the ISA. Wei et al. [19] combined the LWF with a Random Local Search (RLS) and Zang et al. [27] further improved this strategy by a binary search heuristic. Recently, Wei et al. [21] suggested an "Iterative Doubling" Binary Search (IDBS) for their improved skyline based heuristic.

The currently popular state-of-the-art algorithms are GRASP [28], ISA [18], IDBS [21], SRA [25] and IA [26]. Computational results show that IDBS is one of the best algorithms in solving the zero-waste problems because most of the instances are optimally solved. GRASP and IA perform the best for some nonzero-waste problems. However, the time complexity of IDBS, ISA, SRA and IA for decoding one sequence is $O(n^2)$, therefore, they consume too much time to solve larger instances.

The skyline representation of a packing pattern was first proposed by Burke et al. [1] and was used later by many researchers [18,21,25,26]. Burke et al. [1] represented the skyline by a set of "gap", and always selected the lowest available "gap" in each iteration and then found the best rectangle (the rectangle with the largest width and the largest height if there is a tie) fitting into that "gap". Three niche-placement policies were introduced to place the best-fit rectangles. Note that the results of the original BF are independent of the input rectangles sequence, therefore, only three solutions can be generated using different policies. Later on, Imahori and Yagiura [29] gave an efficient implementation with a time complexity of $O(n \log n)$ of the original BF, where n is the number of rectangles. Using the similar framework of the BF, Leung et al. [18] introduced a scoring rule to measure the fitness of each rectangle and five cases are classified. Later, Yang et al. [25] proposed an improved scoring rule and extended the number of cases to eight. Recently, Wei et al. [26] further improved the scoring rules and obtained the best results for most of the instances. We call these best-fit heuristic by Leung et al. [18], Yang et al. [25] and Wei et al. [26] the scoring rule based best-fit (SR-BF). Compared with the original best-fit measure by Burke et al. [1], the scoring rule is more precise. Besides, the solution obtained by the SR-BF is related to the input rectangle sequence, which means that different solutions may be obtained from different sequences. Therefore, the sequence based local search can be used to improve the solution. However, the introduction of such scoring rule decreases the efficiency of the algorithm. The time complexity of all the current SR-BF is $O(n^2)$.

In order to reduce the time complexity of the SR-BF while maintain its effectiveness as well, we propose an improved skyline based heuristic (ISH) for the 2DSP, which can be seen as an improvement of the SR-BF. Similar to the previous SR-BF, the ISH uses the skyline to present the packing pattern. A simple best-fit heuristic is used to construct a solution from a given sequence of rectangles. Finally, a random local search without setting of any parameter is used to further improve the solution. The main difference between the ISH and the previous SR-BF is the best-fit measure in the construction process. The complexity of the scoring rule in the previous SR-BF makes it hard to reduce its time complexity. Our approach simplifies the scoring rule to use the fitness number to select the best-fit rectangle and classifies only four cases. Based on this simplified scoring rule, an efficient implementation of the heuristic with a time complexity of $O(n \log n)$ is proposed. The computational results show that such simplification of the scoring rule does not affect the final solution.

There are three main contributions in this paper. Firstly, we propose a simplified scoring rule to select the best-fit rectangle. Compared with the previous best-fit strategy, our measure is much simpler and easy to implement. Second, we give an efficient implementation with $O(n \log n)$ time complexity of the proposed best-fit heuristic. Finally, our approach is tested on all the existing test sets (to the best of our knowledge), and the results show that our approach outperforms all the existing approaches on most of the instances. The results obtained from the largest instances verify the efficiency of our approach.

The details of our proposed approach are given in Section 2. The computational experiments are reported in Section 3. Finally, Section 4 concludes our paper with some closing remarks.

2. Improved skyline based heuristic for the 2DSP

All of the best-fit heuristic uses the same framework as proposed by Burke et al. [1]. The heuristic uses the skyline to represent the packing pattern. Given a sequence of rectangles, the heuristic repeats the following three steps until all the rectangles are packed: (a) find the lowest and leftmost segment in the skyline; (b) select the rectangle using the best-fit measure; and (c) place the best-fit rectangle at the selected segment. The difference between all the BF heuristics is the method of selecting the best-fit rectangle. In the original BF by Burke et al. [1], the rectangle with the largest width (the largest height if there is a tie) fitting into the gap is selected at each iteration. While in the scoring rule based best-fit by Leung et al. [18], Yang et al. [25] and Wei et al. [26], the rectangle is selected based on the score and several cases are classified for different situations. However, the introduction of a scoring rule increases the time complexity of the BF. This paper proposes a simplified scoring rule and only four cases are classified. Based on this simplification, an efficient implementation with a time complexity of $O(n \log n)$ is given.

In the following section, we will first introduce the best-fit heuristic based on the skyline in Section 2.1. The simplified scoring rule used to select the best-fit rectangle is given in Section 2.2. An efficient implementation is given in Section 2.3 and the detailed technique used is described in Section 2.4. The final approach using a random local search by attempting different sequences is described in Section 2.5.

2.1. Best-fit heuristic based on skyline

We represent a current packing pattern by a rectilinear *skyline*, which is the contour of a set of consecutive vertical bars. Fig. 1 gives an example of a skyline, where each line segment s_j is labeled as j at its left endpoint. Our skyline is the same as those used by Imahori and Yagiura [29] and Wei et al. [21]. According to the definition by Imahori and Yagiura [29] and Wei et al. [21], the skyline can be expressed as a sequence of k horizontal line segments $(s_1, s_2, \ldots, s_k)$ satisfying the following properties: (1) the y-coordinate of s_j is different from the y-coordinate of s_{j+1}, $j = 1, \ldots, k-1$; and (2) the x-coordinate of the right endpoint of s_j is the same as the x-coordinate of the left endpoint of s_{j+1}, $j = 1, \ldots, k-1$. The initial empty packing pattern is represented

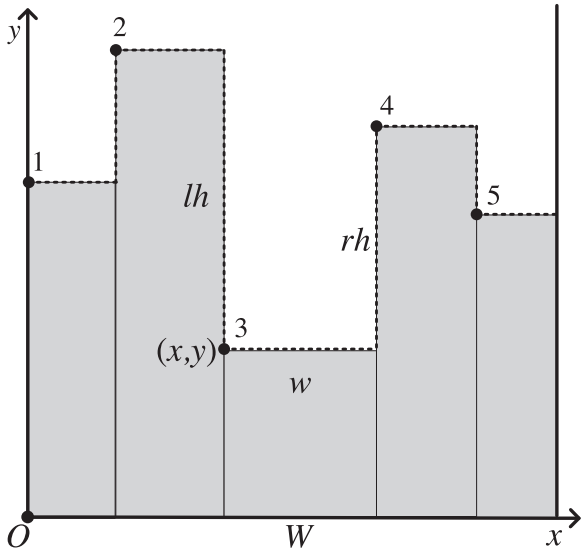

Fig. 1. Example of skyline.

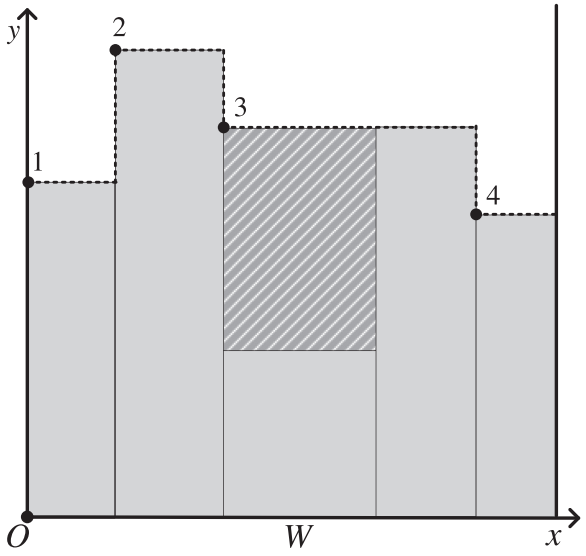

Fig. 2. Update skyline if no rectangle can be fitted into the most bottom-left segment.

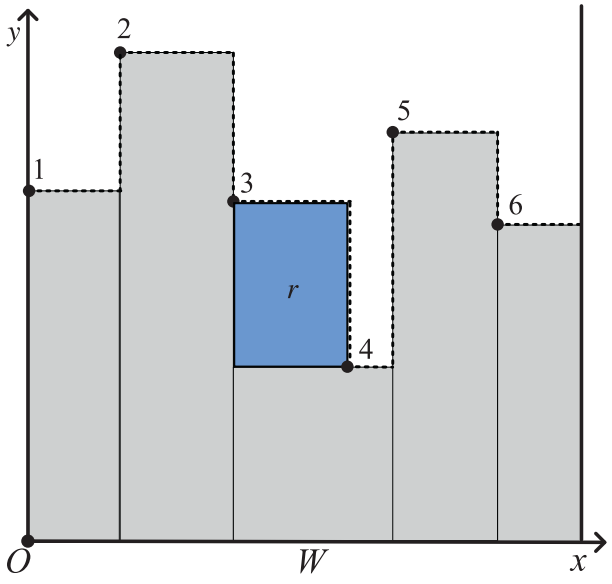

Fig. 3. Update skyline after placing r at the most bottom-left segment.

by a single line segment corresponding to the bottom of the sheet. We use the coordinate of the left endpoint to represent the coordinate of a segment.

Each segment s_j is described by the following attributes:

- x, y: the coordinates of the left corner of s_j;
- w: the width of s_j;
- lh: the height of the vertical segment at the left corner of s_j, i.e., the y-coordinate of s_{j-1} minus the y-coordinate of s_j;
- rh: the height of the vertical segment at the right corner of s_j, i.e., y-coordinate of s_{j+1} minus the y-coordinate of s_j

Both the lh of the first segment s_1 and rh of the last segment s_k are set to infinite.

Our heuristic always selects the most bottom-left segment s as the candidate segment to place a rectangle. If there is no rectangle that can fit into s, that is, the width of all the unpacked rectangles is greater than the width of s, s is raised up to the height of its lower neighbor. In Fig. 1, s_3 is selected as the candidate segment, and if there is no rectangles that fits into s_3, the skyline will be updated as shown in Fig. 2. The shaded area in Fig. 2 is regarded as waste and will not be used in a later stage. If there is more than one rectangle that can fit into s, we select the best-fit rectangle r using our evaluation function. The rectangle is placed either with its bottom-left corner at the left point of s or with its bottom-right corner at the right point of s. After r is placed at s, the skyline will be updated. A new segment is instantiated, corresponding to the top edge of r, and the existing segments affected are updated. Fig. 3 gives the updated skyline after placing a rectangle r at s_3.

Given a sequence of rectangles R and the width of the strip W, the `BestFitPack` constructs solutions by placing the rectangle one after the other, and the final height of the used strip is returned at last. Our best-fit heuristic `BestFitPack` is given as Algorithm 1. In each step, it firstly selects the most bottom-left segment s from the skyline, then picks the best-fit rectangle using our evaluation function (Section 2.2), and places that rectangle at s. The skyline is updated after this placement. The above step is repeated until all the rectangles are packed.

2.2. Scoring rules

Assuming the selected segment is s, we use the *fitness number* to measure the fitness of a placement. Note that r can be placed at either the left or the right corner of s, we will try both placements

Algorithm 1 BestFitPack Procedure.

BestFitPack(R, W)

```
1  Initialize the skyline
2  while not all the rectangles are placed
3      Let s be the bottom-left segment in the skyline
4      Select the best-fit rectangle r from R that can be
       placed at s
5      if r is not found
6          Remove s from the skyline and update the
           skyline
7      else
8          Place r at s and remove r from R
9          Update the skyline
10 return the used height of the strip
```

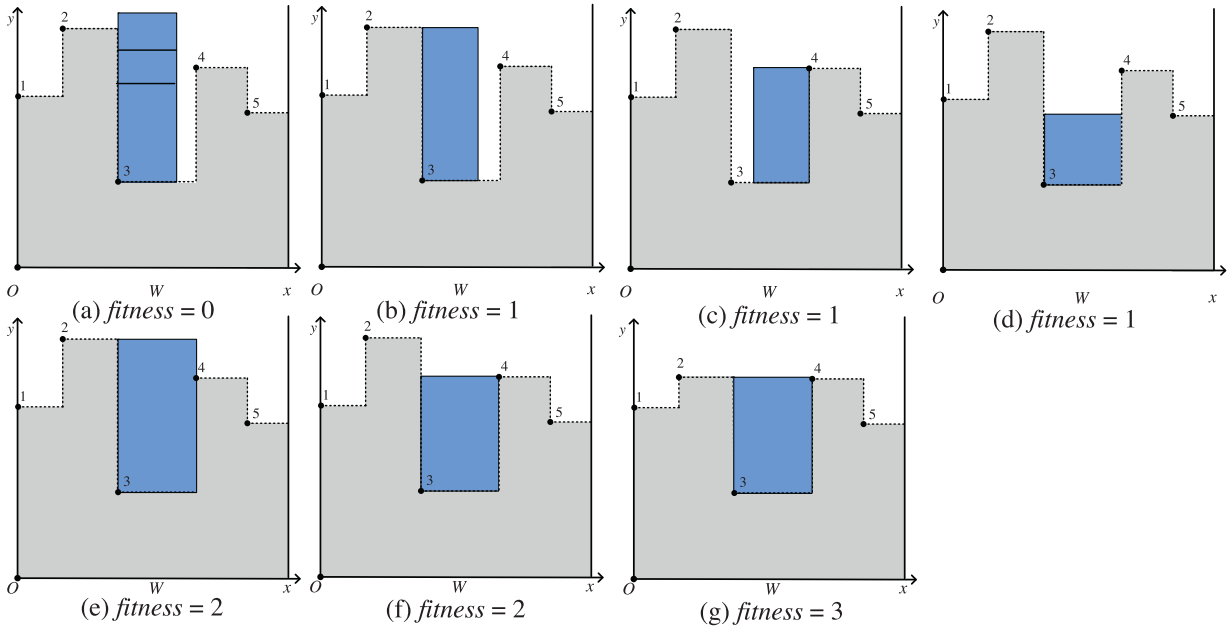

Fig. 4. Fitness number.

and select the best one to represent the *fitness number* of a rectangle r. The fitness number of a placement is the number of sides of the rectangle that exactly matches the segments it is touching in the skyline. The left (resp. right) side of a rectangle is an exact match if its height is equal to $s.lh$ (resp. $s.rh$). The bottom side of a rectangle is an exact match if its width is equal to $s.w$. The fitness number can be either 0, 1, 2 and 3 as shown in Fig. 4.

In our heuristic, we select the rectangle with the *maximum fitness number* and prefer the first unpacked rectangle if there is a tie. We prefer to place that rectangle besides the tallest neighbor (left corner if they are same) if the fitness of both placements is the same. The reason is that we prefer placements resulting in a "smoother" skyline containing fewer line segments, which is more likely to allow the placement of the larger rectangles without producing wasted space.

The fitness number has been used by Wei et al. [21] for the 2D rectangle packing problem. Compared to the scoring rule used by Leung et al. [18], Yang et al. [25] and Wei et al. [26], the fitness number is much simpler and only four cases can be classified based on this number.

2.3. Efficient implementation of the rectangle selection

The BestFitPack repeatedly selects the most bottom-left segment and the best-fit rectangle until all the rectangles are placed. The efficiency of the BestFitPack depends on the time complexity of these two operators.

Same as the method provided by Imahori and Yagiura [29], we use a heap and a doubly linked list to store the skyline. The heap stores the line segments using their y-coordinates as keys and the segment with smaller y-coordinate (with a smaller x coordinates it there is a tie) has a higher priority. As a result, the most bottom-left segment can be found at the root node of the heap. The doubly linked list stores the segments by their x-coordinates. As proved by Imahori and Yagiura [29], all skyline dependent operations can be finished in $O(\log n)$ by using these two data structures.

To select the best-fit rectangle, Imahori and Yagiura [29] and Verstichel et al. [17] used a balanced tree to store the remaining rectangles. In the original BF by Burke et al. [1], the best-fit rectangle is the one with the largest width (the largest height if there is a tie) fitting into the gap. However, the BestFitPack selects the rectangle with the maximum fitness number (selects the first rectangle if there is a tie). Therefore, the balanced tree used by Imahori and Yagiura [29] cannot be applied to the BestFitPack.

Note that the value of fitness number can be 0, 1, 2 and 3. Instead of checking each rectangle one by one, we check the fitness number from largest to smallest to find the best-fit rectangle. In the following, for any rectangle r, we use $r.w$, $r.h$ and $r.i$ to denote the width, height and the position index in the input sequence R, respectively.

Only if $s.lh = s.rh$ and there is a rectangle r with $r.w = s.w$ and $r.h = s.lh$ (or $s.rh$), the fitness can be 3 (case g in Fig. 4). In order to find such a rectangle quickly, we do the following preliminary preparation at the beginning. We copy the input sequence R into another sequence R^1. For each rectangle r, we assign a key value $r.key$ which is computed as $r.h \times (W + 1) + r.w$. Therefore, rectangles with different size have different key values. We then sort R^1 by the key value in increasing order (by the index in increasing order if there is a tie). At each iteration, in order to find the rectangle with given size $s.w \times s.lh$, we only need to find the first rectangle with the corresponding key value; this can be achieved in $O(\log n)$ time complexity by using the binary search. So the time complexity for each iteration is $O(\log n)$.

If $s.lh \neq s.rh$, we check whether there is any rectangle with fitness 2. Only if there is a rectangle r with $r.w = s.w$ and $r.h = s.lh$ (or $r.h = s.rh$), the fitness can be 2 (case e and f in Fig. 4). We can use the same strategy as we use to find the rectangle with fitness 3. So the time complexity is also $O(\log n)$.

If there is no rectangle with fitness equal to 2 or 3, we then check whether there is any rectangle with fitness 1. Only if $r.w = s.w$ or $r.h = s.lh$ or $r.h = s.rh$ (case d, b, and c in Fig. 4), the fitness can be 1. We need to find the first rectangle (the smallest $r.i$) if there is a tie.

In order to find the first rectangle with width $s.w$, we firstly copy the input sequence R into another sequence R^2, then sort R^2 by width, and by index ($r.i$) in increasing order. By using the binary search in R^2, the first rectangle (in R^2) with width $s.w$ can be found in $O(\log n)$ time complexity.

For the case $r.h = s.lh$, the width of the selected rectangle must be smaller or equal to that of the segment, i.e., $r.w \leq s.w$. We first

copy the input sequence R into another sequence R^3, then sort R^3 by height (by width if there is a tie) in increasing order. By using the binary search in R^3, we can find the rectangle with smallest position c (in R^3) whose height is equal to $s.lh$. The rectangle with the largest position d (in R^3) whose height is equal to $s.lh$ and width is not larger than $s.w$ can be found using the similar method. So all the rectangles with height equal to $s.lh$ that can be placed at s are in the range from c to d. The next problem is to find the rectangle whose index $r.i$ is the smallest. This is the classical range minimum query [30] problem and can be solved in $O(\log n)$ time complexity by using the segment tree [31]. The details can be found in Section 2.4. The rectangle with $r.h = s.rh$ can be found by the similar method.

If there is no rectangle with fitness ≥ 1, we need to find the first unpacked rectangle that can be placed at s, more specifically, the first rectangle r with $r.w \leq s.w$ in R. We first copy the input sequence R into another sequence R^4, then sort R^4 by width (by $r.i$ if there is a tie) in increasing order. By using the binary search in R^4, we can find the rectangle with largest position e (in R^4) whose width is not larger than $s.w$. So all the rectangles with a width in the range from 1 to e can be placed at s. The next problem is to find the rectangle whose index $r.i$ is the smallest. This is also the classical range minimum query problem that can be solved in $O(\log n)$ time complexity.

The building of all the auxiliary sequences and the segment tree can be done in $O(n \log n)$ time complexity. At each iteration, the most bottom-left segment can be obtained from the heap in constant time and the best-fit rectangle can be found in $O(\log n)$ time complexity. Once the best-fit rectangle is placed, we need to remove this rectangle from all the auxiliary sequences and the segment tree. As the sorting rule for the different auxiliary sequences is not the same, the position of the best-fit rectangle may vary from sequence to sequence. To remove the best-fit rectangle from all the sequences and the segment tree, we also need auxiliary information to store the position of each rectangle in the auxiliary sequence and the segment tree. By using this information, the position of the best-fit rectangle in each sequence can be found in $O(1)$ time. Thus, removing the rectangle from the auxiliary sequence can be done in $O(1)$ time by just labeling this rectangle as deleted and we need to skip the deleted rectangle while querying this sequence next time. As we always select the first rectangle, these removed rectangles do not affect the time complexity of the query for each iteration. Removing the rectangle from the segment tree can be done in $O(\log n)$ time complexity (the details are given at Section 2.4.3).

After placing the best-fit rectangle on the selected segment, we also need to update the data structures heap and the doubly linked list. Updating the doubly linked list takes constant time, and updating the heap takes time proportional to the height of the heap, which is $O(\log n)$. Thus, the overall time complexity of the BestFitPack is $O(n \log n)$.

2.4. Solve the range minimum query by the segment tree

In our problem, we need to find the rectangle with the smallest index $r.i$ from the given sequence R within the range $[c, d]$. If we copy the entire index $r.i$ into another array I, the problem is to find the minimum number from I within the given range $[c, d]$, which is the Range Minimum Query (RMQ) problem. The RMQ is the problem of finding the smallest element (or the position of the smallest element) in a contiguous subsequence of a list of items. The segment tree [31] can be used to solve the RMQ efficiently. A segment tree, which is based upon the divide-and-conquer paradigm, can be thought of as a tree of intervals of an underlying array. By using the segment tree, the queries on the ranges of the array as well as the modifications to the array's elements can be efficiently performed.

There are three fundamental operations on the segment tree.

2.4.1. Construction of the segment tree

We construct the segment tree top-down recursively. First, we fill in the root node representing the total range $[1, m]$ (m is the size of I) and its two children, and then recursively call the construction process on each of two children, and so on. For any node representing a range $[a, b]$, the range of its left and right children is set to $[a, \lfloor (a+b)/2 \rfloor]$ and $[\lfloor (a+b)/2 \rfloor + 1, b]$, respectively. Each node contains another value representing the minimum value in I between the corresponding range. If the range only contains one item, this node is a leaf node, its value is set to the value of this item. For other nodes, after the two recursive calls made on each child return, that node's value is set to the minimum of the values stored in each child. As the height of the final tree is $\log n + 1$, the construction of the segment tree can be finished in $O(n \log n)$ time complexity.

2.4.2. Query on the segment tree

In our problem, we need to query the minimum value of the items within range $[c, d]$ in I. We query from the root node recursively. For each node representing the range $[a, b]$, three steps are processed as follows (min_l and min_r are initialized to ∞):

1. $c \leq \lfloor (a+b)/2 \rfloor$ (the left child intersects with the query range): recursively call the query process on its left child. Set min_l to the returned value of the query on the left child.
2. $d \geq \lfloor (a+b)/2 \rfloor + 1$ (the right child intersects with the query range): recursively call the query process on its right child. Set min_r to the returned value of the query on its right child.
3. Return the minimal value of min_l and min_r.

The time complexity of the query is the same as the height of the tree, which is $O(\log n)$.

2.4.3. Update on the segment tree

Once the selected rectangle is placed at the segment, we need to remove this rectangle from the segment tree. Assuming the position of the selected rectangle is c, we update the segment tree from the root node recursively as follows. If the node is the leaf node, we set the value of this node to ∞. Otherwise, we call the update process recursively on the child whose range contains c. After this call returns, the value of this node is set to the minimum of the values stored at its children. The time complexity of the update is also $O(\log n)$.

2.5. Random local search for the 2DSP

In order to test the performance of the proposed heuristic, we adapted the RandomizedImprovement proposed by Wei et al. [26] to try different sequences to improve the solution. The adapted procedure RandomLS given as Algorithm 2 is a random local search. It uses a rectangle set R and the width of the strip W as input. The best found height is returned at last.

RandomLS is a sequence based random local search procedure. It firstly uses four sorting rules to generate an initial sequence (line 3) and the BestFitPack is called on each initial sequence. We then use the results to resort the sorting rules.

Thereafter, a random local search is used to improve the solution. It firstly uses four sorting rules to generate the initial sequence (line 8). At each iteration, a new sequence R^* is generated by swapping two randomly selected rectangles from R, and the BestFitPack is called on R^*. Only if the solution of R^* is not worse than that of R, R is replaced by R^*; otherwise, R^* is abandoned. The above iteration is repeated n times, where n is the number of

Table 1
Data set characteristics.

Data source	Data set	#Inst.	n	W	H^*
[11]	C	21	16–197	20–160	15–240
[32]	NT(n)	35	17–199	200	200
	NT(t)	35	17–199	200	200
[1]	n	13	10–3152	30–640	40–960
[33]	CX	7	50–15,000	400	600
[29]	IY	170	$2^4 - 2^{20}$	20,200–5,171,200	20,200–5,171,200
[34]	ngcut	12	7–22	10–30	Unknow
[35]	gcut	13	10–50	250-3000	Unknow
[36]	cgcut	3	16–60	10–70	Unknow
[37]	beng	10	20–200	25–40	Unknow
[38]	bwmv	300	20–100	10–100	Unknow
[39]	bwmv	200	20–100	10–100	Unknow
[40]	Nice	6	25–1000	1000	Unknow
	Path	6	25–1000	1000	Unknow
[41]	ZDF	16	580–75,032	33–5172	Unknow
[42]	AH	360	1000	1000	Unknow

Table 2
Computational environments.

Algo	CPU	RAM	Running times	Time limit
GRASP	Intel Xeon CPU E5405 2.00 GHz	1.99 GB	10	60 s
IDBS	Intel Xeon E5430 2.66 GHz (QuadCore)	8 GB	10	60 s
IA	Intel Xeon CPU E5405 2.00 GHz	1.99 GB	10	60 s
BF	Intel Pentium 4 3.2 GHz	1 GB	1	–
OT3W	Intel Core(TM)2 Duo CPU E8600 (3.33 GHz)	8GB	1	–
ISH	Intel(R) Xeon(R) CPU E5645 2.40 GHz	3.49 GB	10	60 s

Algorithm 2 RandomLS Procedure.

RandomLS(R, W)

```
1  n ← number of rectangles in R
2  for each sorting rule
3      sort R using the sorting rule
4      h ← BestFitPack(R, W)
5  sort the sorting rule in increasing order of the height
   returned by BestFitPack
6  while the time limit is not exceeded
7      for each sorting rule
8          sort R using the sorting rule
9          h ← BestFitPack(R, W)
10         for i ← 1 to n
11             generate sequence R* from R by
                 randomly swapping two rectangles
12             h* ← BestFitPack(R*, W)
13             if h* ≤ h
14                 h ← h*; R ← R*
15 return the minimum found height
```

rectangles in R. At any point in the process, if the height returned by the BestFitPack is equal to the lower bound, then the procedure halts with the found height.

The following four sorting rules are employed to generate the initial sequences for our RandomLS, which are the same as those used by [26]:

1. Sort by area in decreasing order;
2. Sort by height in decreasing order;
3. Sort by width in decreasing order;
4. Sort by perimeter in decreasing order.

The only difference between RandomLS and RandomizedImprovement is the sorting rule. The RandomLS first sorts the sorting rules based on the results and the sorting rules are tried in this order, while RandomizedImprovement always selects one sorting rule from two with best results.

3. Computational experiments

Our improved skyline heuristic (ISH) was implemented as a sequential algorithm using the Visual C++ 6.0 and all experiments were conducted on a Windows computer with Intel(R) Xeon(R) CPU E5645 2.40 GHz and 3.49 GB RAM.

3.1. Benchmark test data

In order to evaluate the performance of our algorithms, we used a wide range of benchmark test instances for the 2D packing from the literature. Some of the test sets were generated by cutting a large sheet into several pieces to form the rectangles, so a perfect optimal solution with 100% area utilization is known. For the remaining test sets, the optimal solution is unknown.

The first test set with known perfect optimal solutions include the data set C, NT(n), NT(t), n, CX and IY. The second test set was not produced by cutting the sheet into pieces and the instances contained do not have known perfect optimal solutions. The second test set consists of 9 data sets, namely ngcut, gcut, cgcut, beng, bwmv, Nice, Path, ZDF and AH.

The characteristics of these data sets are summarized in Table 1. The details of n, W and the known optimal strip height H^* for all the data sets except IY can be found in Wei et al. [21] and Leung et al. [18]. The details of the data set IY can be found in Imahori and Yagiura [29]. It is noted that the data sets CX, ZDF and IY include some extra-large problem instances with $n > 10,000$.

3.2. Results for 2DSP

We ran ISH on each instance 10 times with different seeds set from 1 to 10. The time limit for each run on each instance is set

Table 3
Comparison of average gap of GRASP, IDBS, IA and ISH.

Algo.	C	n	NT (n)	NT (t)	CX	2sp	bwmv	Nice	Path	ZDF	AH	Total
GRASP	0.95	0.95	2.40	2.24	0.88	**2.68**	1.77	1.84	1.68	4.22	–	1.96
IDBS	**0.20**	**0.10**	1.57	1.59	0.46	3.06	2.00	0.56	0.72	2.42	1.00	1.24
IA	0.45	0.13	1.32	1.19	**0.34**	3.07	1.47	0.56	0.40	1.43	0.94	1.05
ISH	0.43	0.15	**1.11**	**0.94**	0.40	2.78	**1.43**	**0.53**	**0.37**	**0.52**	**0.87**	**0.87**

Table 4
Computational results on data set IY.

Instance			avg.gap(100%)				best.gap(100%)		t(s)			
	n	W(LB)	BF	OT3W	IA	ISH	IA	ISH	BF	OT3W	IA	ISH
i4	2^4	20,200	9.42	7.15	**0**	**0**	**0**	**0**	0	0.00	0	0
i5	2^5	28,567	13.55	7.13	**0**	**0**	**0**	**0**	0	0.00	0	1
i6	2^6	40,400	10.35	6.5	2.27	**2.11**	2.06	**1.84**	0	0.00	60	57
i7	2^7	57,134	6.96	4.41	**2.32**	2.40	2.26	**2.21**	0	0.02	60	60
i8	2^8	80,800	5.76	4.18	**2.12**	2.36	**2.07**	2.19	0	0.03	60	60
i9	2^9	114,268	4.5	3.13	**2.07**	2.12	2.07	**1.94**	0	0.07	60	60
i10	2^{10}	161,600	3.97	2.79	1.8	**1.70**	1.8	**1.53**	0	0.13	60	60
i11	2^{11}	228,536	2.55	2.01	1.59	**1.27**	1.59	**1.09**	0.01	0.18	60	60
i12	2^{12}	323,200	2.01	1.61	**0.9**	0.92	0.9	**0.82**	0.02	0.56	60	60
i13	2^{13}	457,073	1.41	1.22	0.87	**0.72**	0.87	**0.60**	0.04	1.01	60	60
i14	2^{14}	646,400	1.11	0.9	**0.62**	0.73	**0.62**	0.70	0.08	2.09	61	60
i15	2^{15}	914,147	0.89	0.66	0.46	**0.43**	0.46	**0.41**	0.18	3.80	66	60
i16	2^{16}	1,292,800	0.64	0.51	0.49	**0.29**	0.49	**0.28**	0.33	10.52	65	60
i17	2^{17}	1,828,295	0.54	0.4	0.25	**0.11**	0.25	**0.10**	0.85	28.71	303	61
i18	2^{18}	2,585,600	0.47	0.29	0.03	**0.04**	**0.03**	0.04	1.92	87.90	1327	62
i19	2^{19}	3,656,590	0.38	0.22	0.01	**0.01**	**0.01**	**0.01**	3.75	264.28	5003	64
i20	2^{20}	5,171,200	0.35	0.19	0.01	**0.004**	0.01	**0.003**	7.69	744.48	18,409	65
Avg.			3.82	2.55	0.93	**0.89**	0.91	**0.81**	0.87	67	1513	53

to 60 s. For each instance, assuming the height found in one run is H, and the lower bound is LB, the gap is defined as $gap = 100 \times (H - LB)/LB$, *avg.gap* is the average value of *gap* over 10 runs and *best.gap* is the best value of *gap* over 10 runs of each instance.

Table 3 gives the comparison on the average gap of GRASP [28], IDBS [21], IA [26] and ISH on all the instances, where the columns show the average gap for all data sets and the best results are highlighted in bold. The computational results of GRASP were taken from [18]. The results of IDBS were provided by the author of [21], and the results of other algorithms were taken from their original paper. The data set 2sp includes four subsets, which are cgcut, gcut, ngcut and beng. Since GRASP and IDBS found feasible solutions for only 11 and 10 out of 16 ZDF instances within the time limit, their corresponding *avg.gap* values are the average of the instances of feasible solutions found. The computational environments of all these approaches are given as Table 2.

The results given in Table 3 show that GRASP is the best algorithm for the data set 2sp, IDBS is the best one for the data sets C and n, and IA is the best one for CX. While ISH is the best algorithm for most sets, including NT(n), NT(t), bwmv, Nice, Path, ZDF and AH. Therefore, it can be concluded that the ISH performs better than all existing algorithms on most of the benchmark instances.

All the instances and the detailed results of each instance obtained by the ISH can be downloaded at http://www.villagerwei.com/. The detailed results of the ISH on each data set are shown in Appendix A.

To test the efficiency of ISH, we tested ISH on the data set IY including large instances with up to 1,048,576 (2^{20}) items. Table 4 provides the comparison results on data set IY, where BF [29] is the best-fit heuristic using an efficient implementation and OT3W [17] is the optimal time three-way heuristic. The results of BF and OT3W were obtained from the corresponding authors. The time complexity of the best-fit heuristic in BF, OT3W and ISH is $O(n \log n)$ while that of the best-fit heuristic in IA is $O(n^2)$. It can be observed that ISH gets the best results among all the compared algorithms. For the largest instance with 1,048,576 items, the time of ISH is 65, which is much less than the time of IA (18,409 s). As the time limit in the process of BestFitPack is not checked, so the cost time slightly exceeds the time limit (60 s). We logged the time passed after each call of BestFitPack on these largest instances, and found that the ISH finished 4 calls of the BestFitPack on these instances within 60 s.

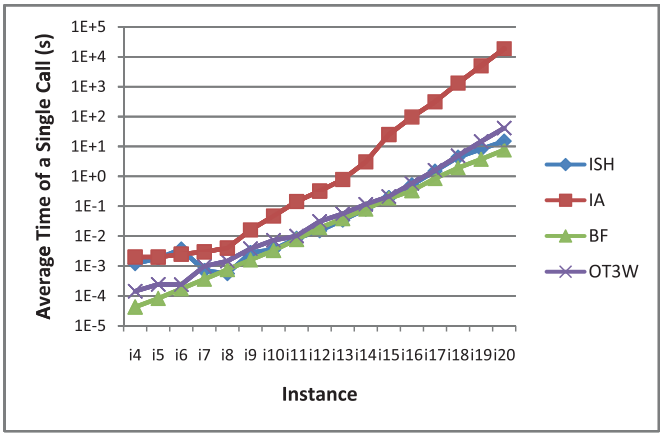

Fig. 5. Average computation time of a single BF call on dataset IY.

To further show the efficiency of ISH, we recorded the time needed for one call of BestFitPack on the instances in IY. The results are shown as Fig. 5. The time of IA is also that of one call of best-fit heuristic. BF includes 3 times and OT3W includes 18 times call of best-fit, so the original time of these approaches is divided by 3 and 18, respectively. For all these instances, ISH can finish one call of BestFitPack within 20 s, which is more than the time required by BF and similar to that of OT3W. The time of ISH is far less than that of IA.

Table 5
Computational results on data set C.

Instance				avg.gap (100%)				best.gap (100%)			
	n	W	LB	IDBS	GRASP	IA	ISH	IDBS	GRASP	IA	ISH
C11	16	20	20	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
C12	17	20	20	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
C13	16	20	20	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
C21	25	40	15	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
C22	25	40	15	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
C23	25	40	15	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
C31	28	60	30	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
C32	29	60	30	**0**	3.3	1.3	**0**	**0**	3.3	**0**	**0**
C33	28	60	30	**0**	**0**	**0**	2	**0**	**0**	**0**	**0**
C41	49	60	60	**0.3**	1.7	1.7	0.7	**0**	1.7	1.7	**0**
C42	49	60	60	**1.5**	1.7	1.7	1.7	**0**	1.7	1.7	1.7
C43	49	60	60	**0**	1.7	**0**	**0**	**0**	1.7	**0**	**0**
C51	73	60	90	**0.2**	1.1	0.8	**0.2**	**0**	1.1	**0**	**0**
C52	73	60	90	**0**	1.1	**0**	**0**	**0**	1.1	**0**	**0**
C53	73	60	90	**0**	1.1	1.0	1.0	**0**	1.1	**0**	**0**
C61	97	80	120	**0.7**	1.7	**0.7**	0.8	**0**	1.7	**0**	0.8
C62	97	80	120	**0**	0.8	0.3	0.6	**0**	0.8	**0**	**0**
C63	97	80	120	**0.7**	1.7	0.8	0.8	**0**	1.7	**0**	0.8
C71	196	160	240	**0.4**	1.7	**0.4**	**0.4**	**0.4**	1.7	**0.4**	**0.4**
C72	197	160	240	**0.4**	1.3	**0.4**	**0.4**	**0.4**	1.3	**0.4**	**0.4**
C73	196	160	240	**0**	1.3	0.4	0.4	**0**	1.3	0.4	0.4
Avg.				**0.20**	0.96	0.45	0.43	**0.04**	0.96	0.22	0.22

4. Conclusions

In this paper, we presented an improved skyline based heuristic for the 2DSP. The packing pattern is represented by the skyline and an simplified best-fit heuristic is used to construct the solution via a given sequence. An efficient implementation of the proposed heuristic with a time complexity of $O(n \log n)$ is given. Finally, a simple random local search without setting of any parameter is used to improve the solution further. The experiment on all the existing test sets (to the best of our knowledge) shows the effectiveness and efficiency of the proposed approach. Compared with the previous best-fit heuristic, our approach is simpler and the time complexity is reduced to $O(n \log n)$. One of the possible further works is to apply this improved heuristic for solving other variants of packing problems.

Acknowledgements

This work was partially supported by the National Natural Science Foundation of China (Grant nos. 71390335 and 71401065), the Natural Science Foundation of Jiangxi Province (Grant no. 20151BAB217006) and China Scholarship Council (Grant no. 201508360050).

Appendix A. The detailed results of ISH on each test set

Tables 5–15 give the detailed results of the ISH on each test set. The best results are highlighted in bold.

Table 6
Computational results on data set n.

Instance				avg.gap (100%)				best.gap (100%)			
	n	W	LB	IDBS	GRASP	IA	ISH	IDBS	GRASP	IA	ISH
N1	10	40	40	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
N2	20	30	50	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
N3	30	30	50	**0**	2	**0**	**0**	**0**	2	**0**	**0**
N4	40	80	80	**0**	1.3	**0**	**0**	**0**	1.3	**0**	**0**
N5	50	100	100	**0**	2	**0**	**0**	**0**	2	**0**	**0**
N6	60	50	100	**0**	1	**0**	0.1	**0**	1	**0**	**0**
N7	70	80	100	**0**	1	**0**	**0**	**0**	1	**0**	**0**
N8	80	100	80	**0**	1.3	1.3	1.3	**0**	1.3	1.3	1.3
N9	100	50	150	**0**	0.7	**0**	0.2	**0**	0.7	**0**	**0**
N10	200	70	150	**0**	0.7	**0**	**0**	**0**	0.7	**0**	**0**
N11	300	70	150	**0**	0.7	**0**	**0**	**0**	0.7	**0**	**0**
N12	500	100	300	**0.2**	1.3	0.3	0.3	**0**	1.3	0.3	0.3
N13	3152	640	960	**0**	0.5	0.1	0.1	**0**	0.5	0.1	0.1
Avg.				**0.01**	0.96	0.13	0.15	**0**	0.96	0.13	0.13

Table 7
Computational results on data set NT (n).

Instance				avg.gap (100%)				best.gap (100%)			
	n	W	LB	IDBS	GRASP	IA	ISH	IDBS	GRASP	IA	ISH
n1a	17	200	200	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
n1b	17	200	200	**0**	4.5	**0**	**0**	**0**	4.5	**0**	**0**
n1c	17	200	200	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
n1d	17	200	200	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
n1e	17	200	200	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
n2a	25	200	200	1.4	3	1.3	**0**	**0**	3	**0**	**0**
n2b	25	200	200	3.1	3	2.6	**0**	3	3	2.5	**0**
n2c	25	200	200	2.1	4	2.2	**1.4**	**0**	4	2	**0**
n2d	25	200	200	1.5	4.5	**0**	0.3	**0**	4.5	**0**	**0**
n2e	25	200	200	0.9	3	2.6	**0**	**0**	3	2.5	**0**
n3a	29	200	200	2.6	4.5	**1.7**	2.7	**0**	4.5	**0**	**0**
n3b	29	200	200	0.4	4	3.5	**3.4**	**0**	4	3.5	3
n3c	29	200	200	2.8	**2.5**	2.9	2.7	**0**	2.5	2.5	2
n3d	29	200	200	1.4	3.5	**0**	**0**	**0**	3.5	**0**	**0**
n3e	29	200	200	**1.8**	3.5	2.5	**1.8**	**0**	3.5	2.5	**0**
n4a	49	200	200	3.2	3	**2**	2.2	2.5	3	**1.5**	2
n4b	49	200	200	3.9	3.5	2.8	**2.5**	3.5	3.5	2.5	**2**
n4c	49	200	200	2.9	2.5	**2**	**2.0**	2.5	2.5	**1.5**	**1.5**
n4d	49	200	200	3.3	3	**1.5**	1.6	2.5	3	**1.5**	**1.5**
n4e	49	200	200	3.3	2.5	1.9	**1.5**	3	2.5	**1.5**	**1.5**
n5a	73	200	200	2.5	2.5	1.8	**1.6**	2	2.5	**1.5**	**1.5**
n5b	73	200	200	2.2	2	1.6	**1.4**	1.5	2	1.5	**1**
n5c	73	200	200	2.8	3	**1.8**	1.9	2	3	**1.5**	**1.5**
n5d	73	200	200	2.3	2	**1.9**	2.0	2	2	**1.5**	**1.5**
n5e	73	200	200	2.5	3	**1.6**	1.7	2	3	1.5	**1**
n6a	97	200	200	1.4	2	1.4	**1.2**	**1**	2	**1**	**1**
n6b	97	200	200	1.3	2	**1**	1.1	1	2	**0.5**	1
n6c	97	200	200	1.2	2	**1**	**1.0**	**0.5**	2	**0.5**	1
n6d	97	200	200	1.8	2.1	**1.2**	1.3	1.5	2	**1**	**1**
n6e	97	200	200	1.7	2	**1.1**	1.3	**1**	2	**1**	**1**
n7a	199	200	200	**0.4**	1	0.5	0.5	**0**	1	0.5	0.5
n7b	199	200	200	**0.2**	1.5	0.5	0.5	**0**	1.5	0.5	0.5
n7c	199	200	200	**0.2**	1.5	0.5	0.5	**0**	1.5	0.5	0.5
n7d	199	200	200	**0.4**	1.5	0.5	0.5	**0**	1.5	0.5	0.5
n7e	199	200	200	**0.5**	1.5	**0.5**	**0.5**	**0.5**	1.5	**0.5**	**0.5**
Avg.				1.59	2.40	1.33	**1.11**	0.91	2.40	1.09	**0.79**

Table 8
Computational results on data set NT (*t*).

Instance				avg.gap (100%)				best.gap (100%)			
	n	*W*	LB	IDBS	GRASP	IA	ISH	IDBS	GRASP	IA	ISH
t1a	17	200	200	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
t1b	17	200	200	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
t1c	17	200	200	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
t1d	17	200	200	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
t1e	17	200	200	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
t2a	25	200	200	**0**	2	**0**	**0**	**0**	2	**0**	**0**
t2b	25	200	200	**0**	4	**0**	**0**	**0**	4	**0**	**0**
t2c	25	200	200	2.5	4	**0**	**0**	**0**	4	**0**	**0**
t2d	25	200	200	1.2	3	2.6	**0**	**0**	3	2.5	**0**
t2e	25	200	200	2.3	3	2.6	**0**	**0**	3	2.5	**0**
t3a	29	200	200	1.5	3.5	**0**	**0**	**0**	3.5	**0**	**0**
t3b	29	200	200	4.5	4.5	3.5	**3.1**	4	4.5	3.5	**0**
t3c	29	200	200	0.0	3	0.5	**0**	**0**	3	**0**	**0**
t3d	29	200	200	2.8	3.5	2	**0**	2	3.5	2	**0**
t3e	29	200	200	**1.2**	4	3.5	3.4	**0**	4	3.5	3
t4a	49	200	200	3.3	2.5	2.4	**2.2**	2.5	2.5	**2**	**2**
t4b	49	200	200	2.8	2.5	**2**	**2.0**	2.5	2.5	**1.5**	**1.5**
t4c	49	200	200	3.2	3	**1.5**	1.6	2.5	3	**1.5**	**1.5**
t4d	49	200	200	3.2	3	**2**	2.2	2.5	3	**1.5**	2
t4e	49	200	200	4.3	3.5	2.8	**2.5**	3.5	3.5	2.5	**2**
t5a	73	200	200	3.1	3	**1.8**	1.9	2.5	3	**1.5**	**1.5**
t5b	73	200	200	2.7	2	**1.4**	1.5	2.5	2	**1**	**1**
t5c	73	200	200	2.8	2.5	1.8	**1.6**	2.5	2.5	**1.5**	**1.5**
t5d	73	200	200	2.4	2	**1.9**	2.0	2	2	**1.5**	**1.5**
t5e	73	200	200	2.2	2	1.6	**1.4**	2	2	1.5	**1**
t6a	97	200	200	1.2	2	**1.1**	**1.1**	**1**	2	**1**	**1**
t6b	97	200	200	1.3	2	**1**	**1.0**	1	2	**0.5**	1
t6c	97	200	200	1.5	2	**1.1**	1.3	**1**	2	**1**	**1**
t6d	97	200	200	1.3	2	**1**	1.1	1	2	**0.5**	1
t6e	97	200	200	1.3	2.5	**1**	1.1	**1**	2.5	**1**	**1**
t7a	199	200	200	**0.5**	1.5	**0.5**	**0.5**	**0.5**	1.5	**0.5**	**0.5**
t7b	199	200	200	**0.5**	1.5	**0.5**	**0.5**	**0.5**	1.5	**0.5**	**0.5**
t7c	199	200	200	**0.5**	2	**0.5**	**0.5**	**0.5**	2	**0.5**	**0.5**
t7d	199	200	200	**0.5**	1	**0.5**	**0.5**	**0.5**	1	**0.5**	**0.5**
t7e	199	200	200	**0.5**	1.5	**0.5**	**0.5**	**0.5**	1.5	**0.5**	**0.5**
Avg.				1.57	2.24	1.19	**0.94**	1.10	2.24	1.04	**0.74**

Table 9
Computational results on data set CX.

Instance				avg.gap (100%)				best.gap (100%)			
	n	*W*	LB	IDBS	GRASP	IA	ISH	IDBS	GRASP	IA	ISH
50cx	50	400	600	**0**	2.2	**0**	0.2	**0**	2.2	**0**	**0**
100cx	100	400	600	3.2	2.8	**2.4**	2.6	2.7	2.8	**1.8**	2.2
500cx	500	400	600	0.05	0.8	**0**	**0**	**0**	0.8	**0**	**0**
1000cx	1000	400	600	**0**	0.3	**0**	**0**	**0**	0.3	**0**	**0**
5000cx	5000	400	600	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
10000cx	10,000	400	600	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
15000cx	15,000	400	600	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Avg.				0.46	0.87	0.34	0.40	0.38	0.87	0.26	0.31

Table 10
Computational results on data set 2sp.

T				avg.gap (100%)			best.gap (100%)		
	n	*W*	LB	GRASP	IA	ISH	GRASP	IA	ISH
cgcut1	16	10	23	**0**	4.3	**0**	**0**	4.3	**0**
cgcut2	23	70	63	**3.2**	**3.2**	**3.2**	**3.2**	**3.2**	**3.2**
cgcut3	62	70	636	3.9	3.9	**3.7**	3.9	3.9	**3.5**
gcut1	10	250	1016	**0**	**0**	**0**	**0**	**0**	**0**
gcut2	20	250	1133	5.1	5	**4.9**	5.1	**4.8**	**4.8**
gcut3	30	250	1803	**0**	**0**	**0**	**0**	**0**	**0**
gcut4	50	250	2934	**2.3**	**2.3**	2.5	**2.3**	**2.2**	**2.2**
gcut5	10	500	1172	**8.6**	**8.6**	**8.6**	**8.6**	**8.6**	**8.6**
gcut6	20	500	2514	4.5	**4.3**	**4.3**	4.5	**4.3**	**4.3**
gcut7	30	500	4641	**1.1**	**1.1**	**1.1**	**1.1**	**1.1**	**1.1**
gcut8	50	500	5703	3.7	**2.9**	**2.9**	3.7	2.8	**2.7**
gcut9	10	1000	2022	**14.6**	**14.6**	**14.6**	**14.6**	**14.6**	**14.6**
gcut10	20	1000	5356	**11.4**	**11.4**	**11.4**	**11.4**	**11.4**	**11.4**
gcut11	30	1000	6537	**5.5**	**5.1**	**5.1**	5.5	**5**	5.1
gcut12	50	1000	12,522	**17.3**	**17.3**	**17.3**	**17.3**	**17.3**	**17.3**
gcut13	32	3000	4772	4.7	4.2	**4.1**	4.7	4.2	**4.0**
ngcut1	10	10	23	**0**	**0**	**0**	**0**	**0**	**0**
ngcut2	17	10	30	**0**	3.3	**0**	**0**	3.3	**0**
ngcut3	21	10	28	**0**	**0**	**0**	**0**	**0**	**0**
ngcut4	7	10	20	**0**	**0**	**0**	**0**	**0**	**0**
ngcut5	14	10	36	**0**	**0**	**0**	**0**	**0**	**0**
ngcut6	15	10	29	**6.9**	**6.9**	**6.9**	**6.9**	**6.9**	**6.9**
ngcut7	8	20	20	**0**	**0**	**0**	**0**	**0**	**0**
ngcut8	13	20	32	**3.1**	6.3	6.3	**3.1**	6.3	6.3
ngcut9	18	20	49	**2**	4.1	**2**	**2**	4.1	**2**
ngcut10	13	30	80	**0**	**0**	**0**	**0**	**0**	**0**
ngcut11	15	30	50	**4**	**4**	**4**	**4**	**4**	**4**
ngcut12	22	30	87	**0**	**0**	**0**	**0**	**0**	**0**
beng1	20	25	30	**0**	3.3	2.7	**0**	3.3	**0**
beng2	40	25	57	**0**	**0**	**0**	**0**	**0**	**0**
beng3	60	25	84	**0**	**0**	**0**	**0**	**0**	**0**
beng4	80	25	107	**0**	**0**	**0**	**0**	**0**	**0**
beng5	100	25	134	**0**	**0**	**0**	**0**	**0**	**0**
beng6	40	40	36	**0**	**0**	**0**	**0**	**0**	**0**
beng7	80	40	67	**0**	**0**	**0**	**0**	**0**	**0**
beng8	120	40	101	**0**	**0**	**0**	**0**	**0**	**0**
beng9	160	40	126	**0**	**0**	**0**	**0**	**0**	**0**
beng10	200	40	156	**0**	**0**	**0**	**0**	**0**	**0**
Avg.				**2.68**	3.06	2.78	**2.68**	3.04	**2.68**

Table 11
Computational results on data set bwmv.

Instance				avg.gap (100%)				best.gap (100%)			
	n	*W*	LB	IDBS	GRASP	IA	ISH	IDBS	GRASP	IA	ISH
C01	20	10	60.3	**1.5**	1.8	1.7	**1.5**	**1.5**	1.8	1.7	**1.5**
	40	10	121.6	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**
	60	10	187.4	**0.6**	**0.6**	**0.6**	**0.6**	**0.6**	**0.6**	**0.6**	**0.6**
	80	10	262.2	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**
	100	10	304.4	0.2	0.2	**0.1**	**0.1**	0.2	0.2	**0.1**	**0.1**
C02	20	30	19.7	**0.3**	0.5	0.6	0.5	**0**	0.5	**0**	0.5
	40	30	39.1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	60	30	60.1	**0**	0.3	**0**	**0**	**0**	0.3	**0**	**0**
	80	30	83.2	**0**	0.1	**0**	**0**	**0**	0.1	**0**	**0**
	100	30	100.5	**0**	0.1	**0**	**0**	0	0.1	**0**	**0**
C03	20	40	157.4	4.4	3.9	4.1	**3.4**	4.2	3.9	3.9	**3.4**
	40	40	328.8	1.8	1.6	**1.3**	**1.3**	1.5	1.6	**1.2**	**1.2**
	60	40	500	1.5	1.3	**1**	**1.0**	1.3	1.3	**1**	**1.0**
	80	40	701.7	1.3	1.1	**1**	**1.0**	1.2	1.1	**1**	**1.0**
	100	40	832.7	0.9	0.9	**0.5**	**0.5**	0.7	0.9	0.5	**0.4**
C04	20	100	61.4	3.4	**3.1**	3.4	3.2	3.1	3.1	**2.9**	3.1
	40	100	123.9	1.4	1.9	**1.2**	**1.2**	1.0	1.9	**1**	**1.0**
	60	100	193.0	1.2	1.9	0.9	**0.8**	0.9	1.9	**0.8**	**0.8**
	80	100	267.2	1.1	1.8	0.7	**0.6**	0.9	1.8	0.5	**0.4**
	100	100	322.0	0.8	1.6	0.5	**0.4**	0.6	1.6	**0.3**	**0.3**
C05	20	100	512.2	4.4	**4.2**	4.3	4.3	4.4	**4.2**	4.3	**4.2**
	40	100	1,053.8	2.3	2	1.9	**1.8**	2.1	2	1.9	**1.8**
	60	100	1,614.0	2.6	2	**1.8**	**1.8**	2.2	2	**1.7**	**1.7**
	80	100	2,268.4	1.4	1	**0.8**	0.9	1.2	1	**0.8**	**0.8**
	100	100	2,617.4	2.4	1.3	1.1	**1.0**	1.7	1.3	**1**	**1.0**
C06	20	10	159.9	5.1	**4.6**	5.3	5.0	4.7	4.6	4.9	**4.5**
	40	10	323.5	3.9	3.1	**2.5**	2.6	3.4	3.1	2.3	**2.2**
	60	10	505.1	3.6	2.9	**2.2**	**2.2**	3.1	2.9	**2**	**2.0**
	80	10	699.7	3.2	2.7	**1.8**	1.9	2.8	2.7	**1.6**	1.7
	100	10	843.8	3.0	2.5	**1.6**	**1.6**	2.6	2.5	**1.4**	**1.4**
C07	20	30	490.4	**2.3**	**2.3**	**2.3**	**2.3**	**2.3**	**2.3**	**2.3**	**2.3**
	40	30	1,049.7	**0.9**	**0.9**	**0.9**	**0.9**	**0.9**	**0.9**	**0.9**	**0.9**
	60	30	1,515.9	**1.0**	**0.9**	**0.9**	**0.9**	**0.9**	**0.9**	**0.9**	**0.9**
	80	30	2,206.1	0.8	**0.7**	**0.7**	**0.7**	**0.7**	**0.7**	**0.7**	**0.7**
	100	30	2,627.0	0.8	**0.6**	**0.6**	**0.6**	0.7	**0.6**	**0.6**	**0.6**
C08	20	40	434.6	5.5	5.5	5.1	**5.0**	5.4	5.5	5	**4.8**
	40	40	922.0	4.2	3.5	**2.7**	2.8	3.6	3.5	**2.4**	**2.4**
	60	40	1,360.9	3.8	3.2	2.4	**2.3**	3.2	3.2	**2.1**	**2.1**
	80	40	1,909.3	3.8	3.3	2.3	**2.1**	3.2	3.3	2	**1.9**
	100	40	2,362.8	3.8	3.1	1.9	**1.8**	3.3	3.1	1.7	**1.6**
C09	20	100	1,106.8	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	40	100	2,189.2	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**
	60	100	3,410.4	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	80	100	4,578.6	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**
	100	100	5,430.5	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**
C10	20	100	337.8	4.0	3.8	3.6	**3.5**	3.9	3.8	3.5	**3.4**
	40	100	642.8	4.0	3.4	**2.9**	**2.9**	3.6	3.4	**2 .8**	**2.8**
	60	100	911.1	3.8	2.6	**2.1**	**2.1**	3.2	2.6	2	**1.9**
	80	100	1,177.6	3.9	2.7	**2**	**2.0**	3.3	2.7	**1.8**	**1.8**
	100	100	1,476.5	3.8	2.4	**1.6**	**1.6**	3.3	2.4	1.5	**1.4**
Avg.				1.99	1.77	1.47	**1.43**	1.76	1.77	1.37	**1.34**

Table 12
Computational results on data set Nice.

Instance				avg.gap (100%)				best.gap (100%)			
	n	*W*	LB	IDBS	GRASP	IA	ISH	IDBS	GRASP	IA	ISH
Nice1	25	1000	1000	4.0	**3.4**	**3.4**	3.6	3.6	3.4	**2.8**	3.5
Nice2	50	1000	1001	**2.7**	4.6	2.9	3.5	**2.1**	4.6	2.6	3.4
Nice3	100	1000	1001	**2.3**	4	2.5	3.1	**2.0**	4	2.3	2.6
Nice4	200	1000	1001	**1.8**	3.6	1.9	2.3	**1.7**	3.6	**1.7**	2.2
Nice5	500	1000	1000	0.9	2.4	**0.6**	0.7	0.8	2.4	**0.6**	0.7
Nice6	1000	1000	999	0.5	2.1	0.5	**0.4**	0.4	2.1	0.5	**0.3**
Nice1t	1000	1000	1001	0.5	2.5	0.9	**0.4**	0.5	2.5	0.9	**0.3**
	1000	1000	1001	0.4	2.1	**0.3**	**0.3**	0.4	2.1	**0.3**	**0.3**
	1000	1000	1000	0.5	2	0.3	**0.2**	0.4	2	0.3	**0.2**
	1000	1000	1000	0.5	1.9	0.6	**0.3**	0.4	1.9	0.6	**0.3**
	1000	1000	1000	0.5	2.2	**0.3**	**0.3**	0.4	2.2	**0.3**	**0.3**
	1000	1000	1001	0.5	1.9	0.4	**0.3**	0.5	1.9	0.4	**0.3**
	1000	1000	1000	0.3	2.2	**0.2**	**0.2**	0.3	2.2	0.2	**0.1**
	1000	1000	1001	0.5	2	0.5	**0.4**	0.4	2	0.5	**0.3**
	1000	1000	1000	0.5	2.2	**0.3**	0.4	0.5	2.2	**0.3**	**0.3**
	1000	1000	1001	0.5	2.6	**0.3**	**0.3**	0.4	2.6	**0.3**	**0.3**
Nice2t	2000	1000	1001	0.3	1.5	0.3	**0.2**	**0.1**	1.5	0.3	**0.1**
	2000	1000	1001	**0.1**	1.4	0.2	**0.1**	**0.1**	1.4	0.2	**0.1**
	2000	1000	1000	0.2	1.6	0.4	**0.1**	**0.1**	1.6	0.4	**0.1**
	2000	1000	1000	0.3	1.4	0.3	**0.2**	0.3	1.4	0.3	**0.1**
	2000	1000	1000	0.2	1.5	0.2	**0.1**	**0.1**	1.5	0.2	**0.1**
	2000	1000	1000	**0.2**	1.6	**0.2**	**0.2**	0.2	1.6	0.2	**0.1**
	2000	1000	1001	0.2	1.5	0.4	**0.1**	**0.1**	1.5	0.4	**0.1**
	2000	1000	1001	0.2	1.3	0.2	**0.1**	**0.1**	1.3	0.2	**0.1**
	2000	1000	1001	**0.1**	1.5	**0.1**	**0.1**	**0.1**	1.5	**0.1**	**0.1**
	2000	1000	1001	0.3	1.5	0.3	**0.2**	0.2	1.5	0.3	**0.1**
Nice5t	5000	1000	1000	**0.1**	1	0.2	**0.1**	**0.1**	1	0.2	**0.1**
	5000	1000	1001	**0.1**	1	0.2	**0.1**	**0.1**	1	0.2	**0.1**
	5000	1000	1001	**0.0**	0.9	0.1	**0.0**	**0.0**	0.9	0.1	**0.0**
	5000	1000	1000	**0.1**	0.9	0.2	**0.1**	**0.0**	0.9	0.2	0.1
	5000	1000	1001	**0.1**	1	**0.1**	**0.1**	**0.0**	1	0.1	**0.0**
	5000	1000	1000	**0.1**	0.9	0.2	**0.1**	**0.1**	0.9	0.2	**0.1**
	5000	1000	1001	**0.0**	1	0.1	0.1	**0.0**	1	0.1	0.1
	5000	1000	1000	**0.1**	1.1	**0.1**	**0.1**	**0.0**	1.1	0.1	0.1
	5000	1000	1001	**0.1**	0.9	0.2	**0.1**	0.1	0.9	0.2	0.1
	5000	1000	1000	**0.0**	1	0.1	**0.0**	**0.0**	1	0.1	**0.0**
Avg.				0.56	1.84	0.56	**0.53**	0.46	1.84	0.52	**0.47**

Table 13
Computational results on data set Path.

Instance				avg.gap (100%)				best.gap (100%)			
	n	*W*	LB	IDBS	GRASP	IA	ISH	IDBS	GRASP	IA	ISH
Path1	25	1000	1001	4.1	4.1	4.1	**4.0**	4.1	4.1	4.1	**3.3**
Path2	50	1000	1000	**0.1**	1.9	**0.1**	**0.1**	**0.1**	1.9	**0.1**	**0.1**
Path3	100	1000	1000	3.0	2.7	1.9	**1.8**	2.5	2.7	1.8	**1.6**
Path4	200	1000	1002	2.4	2.1	**1.2**	**1.2**	2.1	2.1	1.1	**1.0**
Path5	500	1000	1000	2.1	3.4	1.4	**1.0**	1.9	3.4	1.4	**0.9**
Path6	1000	1000	1002	**0.4**	2.4	0.5	0.5	**0.3**	2.4	0.5	0.5
Path1t	1000	1000	999	0.9	2	**0.2**	**0.2**	0.6	2	**0.2**	**0.2**
	1000	1000	1001	0.6	1.7	**0.3**	**0.3**	0.6	1.7	**0.3**	**0.3**
	1000	1000	1001	1.1	1.7	**0.2**	**0.2**	0.6	1.7	**0.2**	**0.2**
	1000	1000	1000	0.7	1.6	**0.3**	**0.3**	0.5	1.6	**0.3**	**0.3**
	1000	1000	1003	0.8	2.1	**0.2**	**0.2**	0.6	2.1	0.2	**0.1**
	1000	1000	1002	0.9	1.6	0.5	**0.4**	0.8	1.6	0.5	**0.4**
	1000	1000	999	0.6	2	**0.2**	**0.2**	0.6	2	**0.2**	**0.2**
	1000	1000	1000	1.3	2	0.4	**0.3**	1.0	2	0.4	**0.3**
	1000	1000	999	0.9	2	**0.4**	**0.4**	0.6	2	0.4	**0.3**
	1000	1000	1002	1.1	1.6	**0.3**	**0.3**	0.6	1.6	0.3	**0.2**
Path2t	2000	1000	1000	**0.0**	1.5	**0**	**0.0**	0.0	1.5	**0**	**0.0**
	2000	1000	1002	0.4	1.4	**0.2**	**0.2**	0.3	1.4	0.2	**0.1**
	2000	1000	1000	0.3	1.5	**0.1**	**0.1**	**0.1**	1.5	**0.1**	**0.1**
	2000	1000	999	0.6	1.5	0.2	**0.1**	0.3	1.5	0.2	**0.1**
	2000	1000	1002	0.2	1.6	0.1	**0.0**	0.1	1.6	0.1	**0.0**
	2000	1000	1002	0.3	1.4	0.3	**0.2**	0.3	1.4	0.3	**0.1**
	2000	1000	998	0.4	1.3	**0.1**	**0.1**	0.3	1.3	0.1	**0.0**
	2000	1000	998	0.3	1.6	**0.2**	**0.2**	0.1	1.6	0.2	**0.1**
	2000	1000	1001	**0.3**	1.6	**0.3**	**0.3**	0.3	1.6	0.3	**0.2**
	2000	1000	1003	0.6	1.5	**0.1**	**0.1**	0.6	1.5	**0.1**	**0.1**
Path5t	5000	1000	1000	**0.1**	1	**0.1**	**0.1**	0.1	1	0.1	**0.0**
	5000	1000	998	0.2	1.1	**0**	**0.0**	0.1	1.1	**0**	**0.0**
	5000	1000	1000	**0.1**	1.1	**0.1**	**0.1**	**0.1**	1.1	**0.1**	**0.1**
	5000	1000	995	0.2	1.1	0.2	**0.1**	**0.1**	1.1	0.2	**0.1**
	5000	1000	1004	0.1	1.2	**0**	**0.0**	0.1	1.2	**0**	**0.0**
	5000	1000	1000	0.0	0.9	**0**	**0.0**	**0.0**	0.9	**0**	**0.0**
	5000	1000	998	0.1	1.1	**0**	**0.0**	0.1	1.1	**0**	**0.0**
	5000	1000	996	0.2	1.1	**0.1**	**0.1**	**0.1**	1.1	**0.1**	**0.1**
	5000	1000	997	0.1	1	0.2	**0.1**	**0.1**	1	0.2	**0.1**
	5000	1000	1002	0.2	1.1	**0**	**0.0**	0.1	1.1	0	**0.0**
Avg.				0.72	1.68	0.40	**0.37**	0.58	1.68	0.40	**0.31**

Table 14
Computational results on data set ZDF.

Instance				avg.gap (100%)			best.gap (100%)		
	n	*W*	LB	GRASP	IA	ISH	GRASP	IA	ISH
zdf1	580	100	330	0.9	**0**	**0**	0.9	**0**	**0**
zdf2	660	100	357	0.8	**0**	**0**	0.8	**0**	**0**
zdf3	740	100	384	0.8	**0**	**0**	0.8	**0**	**0**
zdf4	820	100	407	0.7	**0**	**0**	0.7	**0**	**0**
zdf5	900	100	434	0.7	**0**	**0**	0.7	**0**	**0**
zdf6	1532	3000	4872	7.8	2.9	**2.8**	7.8	2.6	**2.2**
zdf7	2432	3000	4852	6.4	3.1	**2.7**	6.4	**2.6**	**2.6**
zdf8	2532	3000	5172	7.2	4.2	**0.9**	7.2	4.2	**0.5**
zdf9	5032	3000	5172	5.9	2	**0.4**	5.9	2	**0.0**
zdf10	5064	6000	5172	7.7	3.8	**0.1**	7.7	3.8	**0**
zdf11	7564	6000	5172	7.5	2.4	**0.5**	7.5	2.4	**0.1**
zdf12	10,064	6000	5172	–	1.2	**0.5**	–	1.2	**0.1**
zdf13	15,096	9000	5172	–	1.2	**0.5**	–	1.2	**0.3**
zdf14	25,032	3000	5172	–	2.1	**0**	–	2.1	**0**
zdf15	50,032	3000	5172	–	**0**	**0**	–	**0**	**0**
zdf16	75,032	3000	5172	–	**0**	**0**	–	**0**	**0**
Avg.				–	1.43	**0.52**	–	1.38	**0.36**

Table 15
Computational results on data set AH.

Instance			avg.gap (100%)		best.gap (100%)	
	n	W(LB)	IA	ISH	IA	ISH
AH1	1000	20,200	**1.1**	1.4	**1.1**	1.4
AH2	1000	28,567	**1.4**	**1.4**	**1.3**	**1.3**
AH3	1000	40,400	1.1	**1.0**	1.1	**1.0**
AH4	1000	57,134	0.8	**0.7**	0.8	**0.7**
AH5	1000	80,800	**1.0**	1.1	**1.0**	**1.0**
AH6	1000	114,268	1.1	**0.9**	1.0	**0.8**
AH7	1000	161,600	**0.7**	0.8	**0.7**	**0.7**
AH8	1000	228,536	**0.6**	**0.6**	0.6	**0.5**
AH9	1000	323,200	1.8	**1.4**	1.8	**1.3**
AH10	1000	457,073	0.9	**0.7**	0.9	**0.6**
AH11	1000	646,400	0.6	**0.3**	0.6	**0.3**
AH12	1000	914,147	**0.2**	**0.2**	0.2	**0.1**
Avg.			0.94	**0.87**	0.90	**0.81**

References

[1] Burke EK, Kendall G, Whitwell G. A new placement heuristic for the orthogonal stock-cutting problem. Oper Res 2004;52(4):655–71. doi:10.1287/opre.1040.0109.

[2] Wäscher G, Haußner H, Schumann H. An improved typology of cutting and packing problems. Eur J Oper Res 2007;183(3):1109–30. doi:10.1016/j.ejor.2005.12.047.

[3] Hifi M. Exact algorithms for the guillotine strip cutting/packing problem. Comput Oper Res 1998;25(11):925–40. doi:10.1016/S0305-0548(98)00008-2.

[4] Martello S, Monaci M, Vigo D. An exact approach to the strip-packing problem. INFORMS J Comput 2003;15(3):310–19. doi:10.1287/ijoc.15.3.310.16082.

[5] Kenmochi M, Imamichi T, Nonobe K, Yagiura M, Nagamochi H. Exact algorithms for the two-dimensional strip packing problem with and without rotations. Eur J Oper Res 2009;198(1):73–83. doi:10.1016/j.ejor.2008.08.020.

[6] Boschetti MA, Montaletti L. An exact algorithm for the two-dimensional strip-packing problem. Oper Res 2010;58(6):1774–91. doi:10.1287/opre.1100.0833.

[7] Belov G, Rohling H. LP bounds in an interval-graph algorithm for orthogonal–packing feasibility. Oper Res 2013;61(2):483–97.

[8] Côté J-F, Dell'Amico M, Iori M. Combinatorial benders' cuts for the strip packing problem. Oper Res 2014;62(3):643–61. doi:10.1287/opre.2013.1248.

[9] Baker BS, Coffman EG, Rivest RL. Orthogonal packings in two dimensions. SIAM J Comput 1980;9(4):846–55. http://scitation.aip.org/getabs/servlet/GetabsServlet?prog=normal&id=SMJCAT000009000004000846000001&idtype=cvips&gifs=yes

[10] Chazelle. The bottom-left bin-packing heuristic: an efficient implementation. IEEE Trans Comput 1983;32(8):697–707. doi:10.1109/TC.1983.1676307.

[11] Hopper E, Turton BCH. An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. Eur J Oper Res 2001;128(1):34–57. doi:10.1016/S0377-2217(99)00357-4.

[12] Zhang D, Kang Y, Deng A. A new heuristic recursive algorithm for the strip rectangular packing problem. Comput Oper Res 2006;33(8):2209–17. doi:10.1016/j.cor.2005.01.009.

[13] Cui Y, Yang Y, Cheng X, Song P. A recursive branch-and-bound algorithm for the rectangular guillotine strip packing problem. Comput Oper Res 2008;35(4):1281–91. doi:10.1016/j.cor.2006.08.011.

[14] He K, Jin Y, Huang W. Heuristics for two-dimensional strip packing problem with 90° rotations. Expert Syst Appl 2013;40:5542–50. http://www.sciencedirect.com/science/article/pii/S095741741300242X

[15] Wang Y, Chen L. Two-dimensional residual-space-maximized packing. Expert Syst Appl 2015;42(7):3297–305. doi:10.1016/j.eswa.2014.12.021.

[16] Aşık O, Özcan E. Bidirectional best-fit heuristic for orthogonal rectangular strip packing. Ann Oper Res 2009;172:405–27. doi:10.1007/s10479-009-0642-0.

[17] Verstichel J, Causmaecker PD, Berghe GV. An improved best-fit heuristic for the orthogonal strip packing problem. Int Trans Oper Res 2013;20(5):711–30.

[18] Leung SCH, Zhang D, Mong Sim K. A two-stage intelligent search algorithm for the two-dimensional strip packing problem. Eur J Oper Res 2011;215(1):57–69. doi:10.1016/j.ejor.2011.06.002.

[19] Wei L, Zhang D, Chen Q. A least wasted first heuristic algorithm for the rectangular packing problem. Comput Oper Res 2009;36(5):1608–14. doi:10.1016/j.cor.2008.03.004.

[20] Wu Y-L, Huang W, Lau S-c, Wong CK, Young GH. An effective quasi-human based heuristic for solving the rectangle packing problem. Eur J Oper Res 2002;141(2):341–58. doi:10.1016/S0377-2217(02)00129-7.

[21] Wei L, Oon W-C, Zhu W, Lim A. A skyline heuristic for the 2D rectangular packing and strip packing problems. Eur J Oper Res 2011;215(2):337–46. doi:10.1016/j.ejor.2011.06.022.

[22] Liu D, Teng H. An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles. Eur J Oper Res 1999;112(2):413–20. doi:10.1016/S0377-2217(97)00437-2.

[23] Zhang D, Liu Y, Chen S, Xie X. A meta-heuristic algorithm for the strip rectangular packing problem. In: Wang L, Chen K, Ong, editors. Advances in natural computation. Lecture Notes in Computer Science, 3612. Berlin, Heidelberg: Springer; 2005. p. 1235–41. doi:10.1007/11539902_157. ISBN 978-3-540-28320-1

[24] Burke EK, Kendall G, Whitwell G. A simulated annealing enhancement of the best-fit heuristic for the orthogonal stock-cutting problem. INFORMS J Comput 2009;21(3):505–16. doi:10.1287/ijoc.1080.0306.

[25] Yang S, Han S, Ye W. A simple randomized algorithm for two-dimensional strip packing. Comput Oper Res 2013;40(1):1–8. doi:10.1016/j.cor.2012.05.001.

[26] Wei L, Qin H, Cheang B, Xu X. An efficient intelligent search algorithm for the two-dimensional rectangular strip packing problem. Int Trans Oper Res 2016;23:65–92. doi:10.1111/itor.12138.

[27] Zhang D, Wei L, Leung SCH, Chen Q. A binary search heuristic algorithm based on randomized local search for the rectangular strip-packing problem. INFORMS J Comput 2013;25(2):332–45. doi:10.1287/ijoc.1120.0505.

[28] Alvarez-Valdes R, Parreño F, Tamarit JM. Reactive grasp for the strip-packing problem. Comput Oper Res 2008;35(4):1065–83. doi:10.1016/j.cor.2006.07.004.

[29] Imahori S, Yagiura M. The best-fit heuristic for the rectangular strip packing problem: an efficient implementation and the worst-case approximation ratio. Comput Oper Res 2010;37(2):325–33. doi:10.1016/j.cor.2009.05.008.

[30] Gabow HN, Bentley JL, Tarjan RE. Scaling and related techniques for geometry problems. In: Proceedings of the sixteenth annual ACM symposium on theory of computing. STOC '84. New York, NY, USA: ACM; 1984. p. 135–43. ISBN 0-89791-133-4. doi:10.1145/800057.808675.

[31] Bentley JL. Algorithms for Klee's rectangle problems. Technical Report. Computer Science Department, Carnegie Mellon University; 1977.

[32] Hopper E. Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods, Ph.D. thesis. University of Wales, Cardiff School of Engineering; 2000.

[33] Pinto E, Oliveira JF. Algorithm based on graphs for the non-guillotinable two-dimensional packing problem. In: Proceedings of the second ESICUP meeting, Southampton, United Kingdom; 2005.

[34] Beasley JE. An exact two-dimensional non-guillotine cutting tree search procedure. Oper Res 1985;33(1):49–64. doi:10.2307/170866.

[35] Beasley JE. Algorithms for unconstrained two-dimensional guillotine cutting. J Oper Res Soc 1985;36(4):297–306. doi:10.2307/2582416.

[36] Christofides N, Whitlock C. An algorithm for two-dimensional cutting problems. Oper Res 1977;25(1):30–44. doi:10.2307/169545.

[37] Bengtsson BE. Packing rectangular pieces – a heuristic approach. Comput J 1982;25:253–7.

[38] Berkey JO, Wang PY. Two-dimensional finite bin-packing algorithms. J Oper Res Soc 1987;38(5):423–9. doi:10.2307/2582731.

[39] Martello S, Vigo D. Exact solution of the two-dimensional finite bin packing problem. Manage Sci 1998;44(3):388–99. doi:10.1287/mnsc.44.3.388.

[40] Wang PY, Valenzela CL. Data set generation for rectangular placement problems. Eur J Oper Res 2001;134(2):378–91. doi:10.1016/S0377-2217(00)00263-0.

[41] Leung SCH, Zhang D. A fast layer-based heuristic for non-guillotine strip packing. Expert Syst Appl 2011;38(10):13032–42. http://dx.doi.org/10.1016/j.eswa.2011.04.105.

[42] Bortfeldt A. A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces. Eur J Oper Res 2006;172(3):814–37. doi:10.1016/j.ejor.2004.11.016.