A Skyline-Based Heuristic for the 2D Rectangular Strip Packing Problem

Conference Paper · June 2011

DOI: 10.1007/978-3-642-21827-9_29 · Source: DBLP

3 authors:

Wei Lijun
GuangDong University of Technology
44 PUBLICATIONS **468** CITATIONS
SEE PROFILE

Andrew Lim
National University of Singapore
432 PUBLICATIONS **7,508** CITATIONS
SEE PROFILE

Wenbin Zhu
South China University of Technology
61 PUBLICATIONS **695** CITATIONS
SEE PROFILE

European Journal of Operational Research 215 (2011) 337–346

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

A skyline heuristic for the 2D rectangular packing and strip packing problems

Lijun Wei [a], Wee-Chong Oon [a], Wenbin Zhu [b,c,*], Andrew Lim [a]

[a] Department of Management Sciences, City University of Hong Kong, Tat Chee Ave, Kowloon Tong, Hong Kong
[b] Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong
[c] Department of Logistics and Maritime Studies, Faculty of Business, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong

ARTICLE INFO

Article history:
Received 20 January 2011
Accepted 14 June 2011
Available online 25 June 2011

Keywords:
Cutting and packing
Heuristics
Tabu search

ABSTRACT

In this paper, we propose a greedy heuristic for the 2D rectangular packing problem (2DRP) that represents packings using a skyline; the use of this heuristic in a simple tabu search approach outperforms the best existing approach for the 2DRP on benchmark test cases. We then make use of this 2DRP approach as a subroutine in an "iterative doubling" binary search on the height of the packing to solve the 2D rectangular strip packing problem (2DSP). This approach outperforms all existing approaches on standard benchmark test cases for the 2DSP.

1. Introduction

The 2D rectangular packing problem (2DRP) is a fundamental problem in cutting and packing literature. We are 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 rectangle of dimensions $W \times H$ (which we call the *sheet*) such that the total area of the packed rectangles is maximized. All dimensions are assumed to be integral. We consider two variants of the problem: the *fixed orientation* variant does not allow the rotation of rectangles, while the *rotatable* variant allows the rectangles to be rotated by 90°.

Another important problem in cutting and packing, called the 2D strip packing problem (2DSP), is closely related to the 2DRP. Given a set of n rectangles, the task is to pack all the rectangles into a rectangular strip of width W so as to minimize the height H of the packing. One possible algorithmic solution to the 2DSP is as follows: set a height H and solve the 2DRP for a sheet of dimensions $W \times H$. If all rectangles can be placed, then a 2DSP solution with height H has been found, and the procedure can be repeated after decreasing H; otherwise, increase H. This idea has previously been proposed by Oliveira and Ferreira (1993) and Dowsland (1993).

There are two main contributions in this paper. Firstly, we propose a greedy 2DRP heuristic that places each rectangle according to an evaluation function involving several components, all of which are motivated by observations on the nature of 2D packing problems. We show that by simply using this heuristic on a number of sequences determined by a tabu search approach, we are able to outperform the best existing approach to the 2DRP on benchmark test instances. Secondly, we incorporate this 2DRP solution as a subroutine in an "iterative doubling" binary search on the height of the sheet to solve the 2DSP, which is more efficient than the previously proposed approaches by Oliveira and Ferreira (1993) and Dowsland (1993). This approach outperforms all existing approaches on the 2DSP.

The rest of the paper is organized as follows. In Section 2, we provide a review of the relevant literature on the 2DRP and 2DSP. We present our greedy heuristic for the 2DRP in Section 3, which is employed in the tabu search procedure given in Section 4. We then describe in Section 5 how this procedure is used as a subroutine in an iterative doubling binary search algorithm to solve the 2DSP. Our computational experiments for both the 2DRP and 2DSP are reported in Section 6. Finally, we conclude our article in Section 7 with some closing remarks.

2. Literature review

There are many practical cutting and packing problems in industry, giving rise to many variants. These variants can be classified according to several attributes. A non-exhaustive list includes (1) by *dimensionality*: the rectangles to be packed are usually either two-dimensional or three-dimensional (although the problem can be generalized to higher dimensions). The 2D variants naturally model the cutting of a sheet of stock into smaller parts, while the 3D variants reflect the packing of items into a container; (2) by *shape*: the majority of existing work assumes the items to be packed are rectangular. The study of 2D packing with non-rectangular items is useful in applications like the cutting of bolts of cloth

* Corresponding author at: Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. Tel.: +852 64067667; fax: +852 34420188.
E-mail address: i@zhuwb.com (W. Zhu).

0377-2217/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2011.06.022

into material for clothing; (3) by *rotatability*: the problem may allow or disallow the orthogonal rotation of items. Rotation may be disallowed for applications where the sheet is patterned (such as patterned cloth or wooden planks with grain patterns), or the positioning of advertisements on a newspaper page; (4) by *guillotinability*: many applications require that the solution fulfills some type of guillotine-cut constraint, where the resultant pattern must be producible by a series of cuts parallel to the axes. Examples include the cutting of brittle glass sheets and the use of machinery that performs only guillotine cuts; (5) by *objective*: the problem can be generalized so that the objective is to maximize the total value of the placed rectangles, and the value of each rectangle does not necessarily correspond to its area.

Our research focuses on the 2D case where all items are rectangles and the objective is to maximize the area utilization. We do not handle the guillotine-cut constraint, but we do consider rotatability. Under the improved typology of cutting and packing problems by Wäscher et al. (2007), the 2DRP is a two-dimensional rectangular single large object placement problem (2D-SLOPP), while the 2DSP is a two-dimensional open dimension problem (2D-ODP). We review only the relevant work of this type in the remainder of this section.

Both the 2DRP and 2DSP are NP-hard in the strong sense. Although some exact algorithms have been proposed for the 2DRP (Beasley, 1985b; Lesh et al., 2004) and the 2DSP (Hifi, 1998; Martello et al., 2003; Kenmochi et al., 2009), these problems usually require the use of heuristics in practice to provide a good solution in reasonable time.

Various heuristic algorithms based on different methodologies have been presented for solving these problems. The bottom-left (BL) method by Baker et al. (1980) places each rectangle sequentially by pushing it downwards as much as possible, then leftwards as much as possible. A genetic algorithm based on BL was later devised by Liu and Teng (1999). The bottom-left fill (BLF) method by Chazelle (1983) generalized the BL concept by placing each rectangle in its bottom-leftmost position. Both BL and BLF placed the rectangles in sequence according to decreasing area. Hopper and Turton (2001) proposed executing the BL heuristic on a number of different sequences and selecting the best result, an approach subsequently called bottom-left-decreasing (BLD).

Instead of prioritizing bottom-left placement, another common approach is to select both the rectangle and its position based on some measure of best fit (i.e., a greedy heuristic). Burke et al. (2004) suggested one such measure (BF) involving a number of priority rules; an efficient implementation for BF was later presented by Imahori and Yagiura (2010) along with an analysis of its worst-case approximation ratio. Wu et al. (2002) introduced another measure called Less Flexibility First. More recently, Aşık and Özcan (2009) presented a novel bidirectional best-fit heuristic.

Several metaheuristic approaches have also been tried. For the 2DRP, Huang et al. (2007) devised a backtracking procedure that locates good "corner-occupying actions", which are rectangle placements evaluated by the notion of caving degree. Gonçalves (2007) later proposed a hybrid genetic algorithm based on random keys. The Least Wasted First (LWF) strategy was utilized by Wei et al. (2009) in a randomized random local search for this problem. The best approach on benchmark test data to date is the hybrid simulated annealing (HSA) approach by Leung et al. (2012). It uses a greedy heuristic to find a locally good solution, and then performs a simulated annealing procedure to perturb the solution in order to search other neighborhoods.

For the 2DSP, Zhang et al. (2006) proposed a new heuristic recursive algorithm, and then devised a simulated annealing algorithm based on this strategy (Zhang et al., 2005). Bortfeldt (2006) presented a genetic algorithm called SPGAL that works directly on layouts rather than encoding the solutions. Burke et al. (2009) examined first using the BF heuristic to pack some rectangles, and then applying a simulated annealing metaheuristic for the remaining rectangles. Leung and Zhang (2010) presented a simple but fast heuristic algorithm based on a brick-laying strategy that can solve large-scale problems effectively. For the fixed orientation variant, the greedy randomized adaptive search procedure (GRASP) by Alvarez-Valdes et al. (2008) had the best results at the time of this writing.

3. Heuristic for the 2DRP

In this section, we present a new greedy heuristic for the 2DRP, which we will subsequently use in our 2DSP approach. This deterministic heuristic takes as input a sequence of rectangles, the dimensions $W \times H$ of the sheet, and a controlling parameter *max_spread*.

3.1. Skyline representation of a packing pattern

We represent a current packing pattern by a rectilinear *skyline*, which is the contour of a set of consecutive vertical bars. It 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}, $j = 1, \ldots, k-1$. Fig. 1 gives an example of a skyline, where each line segment s_j is labeled as j at its left endpoint. The initial empty packing pattern is represented by a single line segment corresponding to the bottom of the sheet.

Our heuristic places rectangles one by one. Each rectangle is placed with either its left bottom corner touching a left endpoint or its right bottom corner touching a right endpoint of a line segment s_j in the skyline. The left endpoint of a segment s_j is a candidate position if and only if s_{j-1} is higher than s_j; similarly, the right endpoint of a segment s_j is a candidate position if and only if s_{j+1} is higher than s_j. Note that the left endpoint of s_1 and the right endpoint of s_k are always candidates. The candidate positions of the skyline in Fig. 1 are depicted by dots.

When a rectangle b is placed on a segment s_j, the skyline is updated. This is done in two steps. The first step instantiates a new line segment corresponding to the top edge of b and updates the existing segments affected. There are two cases depending on whether the width of b is greater than s_j; Fig. 2 shows an example of the two cases when a rectangle b is placed on the left endpoint of s_j, along with how the skyline is updated. Note that the shaded area in Fig. 2(d) is considered *wasted space* because our heuristic will never consider placing any rectangles into it.

In the second step, we check each line segment that is lower than both its adjacent segments, which we call a *locally lowest*

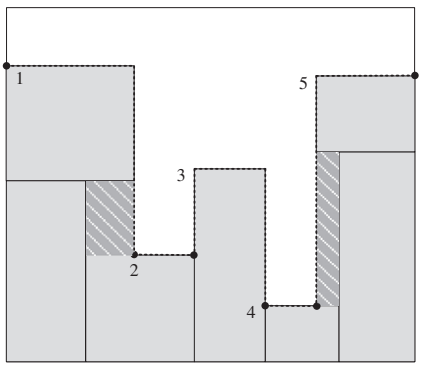


Fig. 1. Example of skyline.

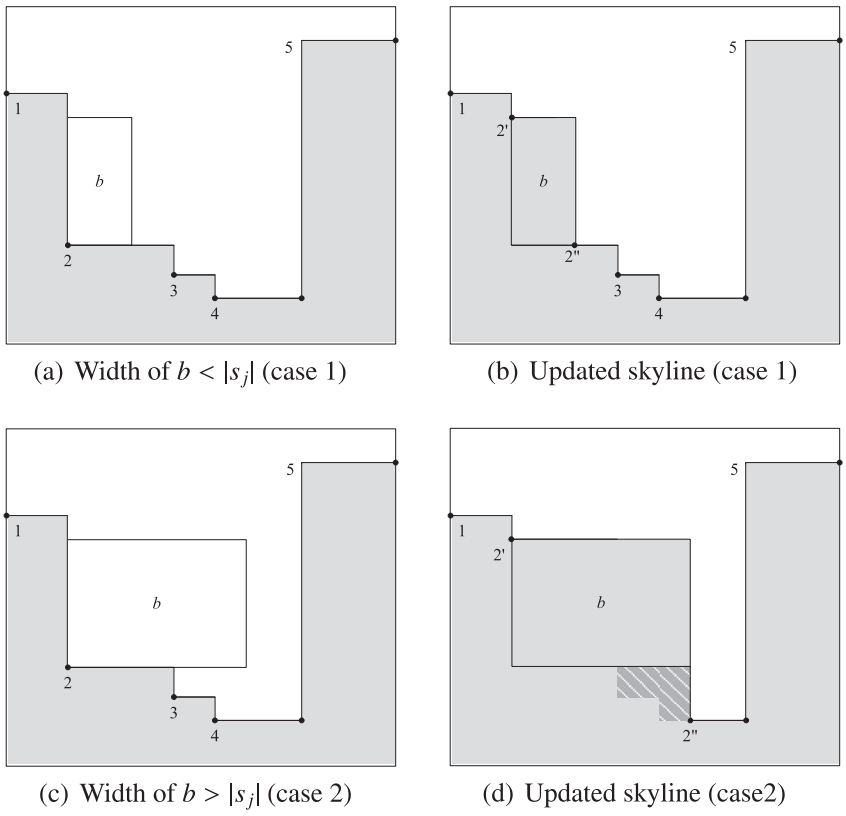


Fig. 2. Updating skyline step 1.

segment (for the first and last segments, we only compare them with their one adjacent segment). If there are no unplaced rectangles that can be placed on that segment, we raise it to the height of its lower adjacent segment and merge them. This is repeated until all line segments are considered. In Fig. 3(a), the rectangle b is placed as shown and the first updating step has been performed. If we find that no unplaced rectangles can be placed on the locally lowest segment s_3, then it is raised and merged with s_4 to form segment $s_{3'}$ (Fig. 3(b)). Similarly, if b was the only remaining rectangle that could have been placed on s_5, then s_5 will also be raised (and merged with $s_{3'}$), as shown in Fig. 3(c). The shaded areas are similarly considered wasted space.

3.2. Placement evaluation

Our heuristic considers all possible placements (i.e., a position-rectangle pair), picks the best one and places the rectangle at that position. This is repeated until no further rectangles can be placed at any position (a failure) or all rectangles have been placed. We first state our evaluation function, and then explain the motivations behind each of its components.

Consider a feasible placement (p,b), where p is a candidate position on line segment s_j. When evaluating a placement, the algorithm examines the resultant skyline after the *first* step of our skyline updating procedure; this is done for reasons of time efficiency. The evaluation function for a given feasible placement (p,b) is then given by the following set of priority rules:

1. (**spread constraint**) The *spread* of a skyline is the difference between the y-coordinates of its highest and lowest line segments. If the spread of the resultant skyline after placing the rectangle at that position is greater than the input parameter *max_spread*, then the placement is considered infeasible and immediately rejected.

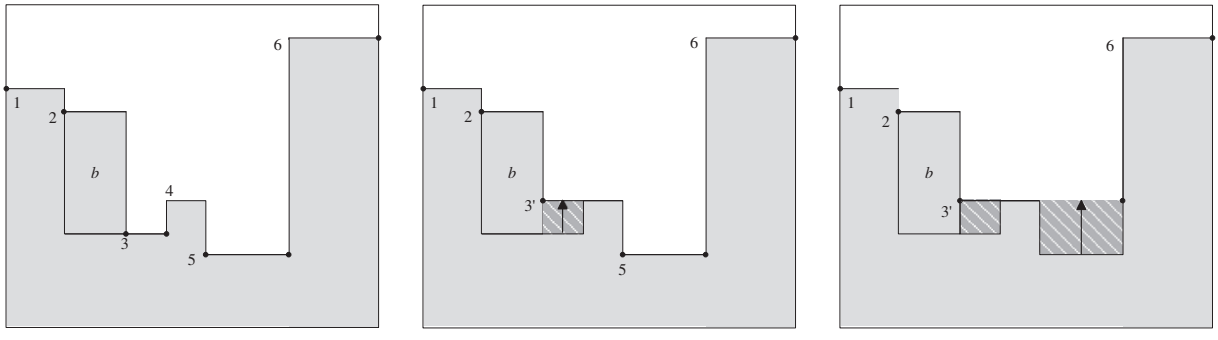
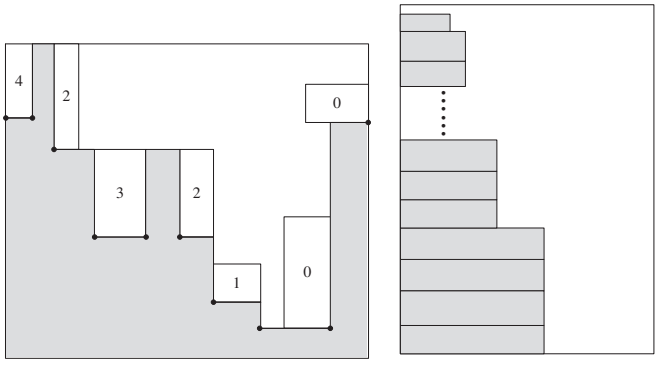


Fig. 3. Updating skyline step 2.

2. (**only fit**) If the rectangle b is the only rectangle remaining that can fit onto p, then (p,b) has the highest priority.
3. (**minimum local waste**) If there are no such cases or more than one such case, we select the placement that minimizes the amount of *wasted local space*. This is calculated as the total volume of wasted space of the four types given in Fig. 4(a)–(d). Let *w_min* and *h_min* be the minimum width and height, respectively, of the remaining rectangles excluding b. The shaded area in case (a) is always considered wasted space. The shaded areas in cases (b) and (c) are considered wasted if the length of the *gap* is less than *w_min*. Finally, the shaded area in case (d) is wasted if the *gap* is less than *h_min*.
4. (**maximum fitness number**) If there is a tie, prefer the placement with highest *fitness number*. The fitness number of a placement is the number of sides of the rectangle that exactly matches the segment it is touching in the skyline. The bottom side of a rectangle is an exact match if its width is equal to the length of s_j. The left (resp. right) side of a rectangle is an exact match if its height is equal to the y-coordinate of s_{j-1} (resp. s_{j+1}) minus the y-coordinate of s_j. Any side of a rectangle that touches the left, right or bottom side of the sheet is not considered an exact match unless it fills all the remaining space on that side. However, when the top edge of a placed rectangle touches the top of the sheet, this is conisidered an exact match. The fitness number can be either 0, 1, 2, 3 or 4, as shown in Fig. 5(a), where the number in each rectangle depicted is its fitness number.
5. (**earliest in sequence**) If there is a tie, prefer the placement involving the earliest rectangle in the input sequence. If the rectangle can be placed at multiple locations, prefer the location with the smallest y-coordinate, then with the smallest x-coordinate.

For the rotatable variant, we try both orientations for each rectangle, and the better orientation determines the evaluation for the placement. If both orientations have identical evaluations based on the above five rules, we prefer the orientation corresponding to the original input.

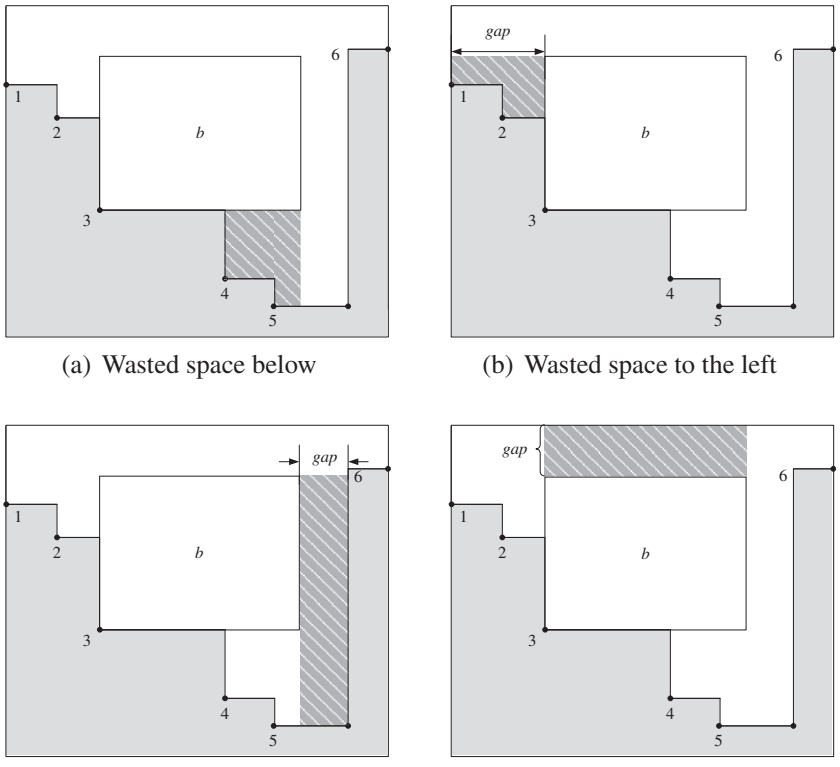


(a) Fitness numbers (b) Tall tower

Fig. 5. Fitness numbers and the tall tower phenomenon.

There are two major motivations for implementing a **spread constraint**. Firstly, without this restriction our greedy heuristic would tend to produce a tall tower of rectangles along the left side until the sheet height H is reached (see Fig. 5(b)). This is a possibly undesirable phenomenon that can be controlled with the spread constraint. Secondly, different values of *max_spread* causes our heuristic to behave differently: a high *max_spread* value allows tall towers, while a low *max_spread* value causes the heuristic to place rectangles layer by layer. By trying a range of *max_spread* values, we can generate more diverse packings, thereby exploring the search space more thoroughly.

Recall that we only perform the first step of the skyline updating routine when performing our evaluation. This detects wasted space in adjacent segments of the types shown in the example in Fig. 4 but does not take other types of wasted space into account



(a) Wasted space below (b) Wasted space to the left

(c) Wasted space to the right (d) Wasted space above

Fig. 4. Wasted local space.

(like the example in Fig. 3(c)). The **only fit** criterion helps to address this omission. It is based on the observation that if the only rectangle b that can be placed on a line segment s_j is placed elsewhere, then the area just above s_j is wasted, which might be large. Hence, it is likely that placing b on segment s_j will reduce the overall amount of wasted space.

The key component of our evaluation function is the **minimum local waste** priority rule. It is motivated by the natural assumption that if the amount of wasted space is minimized at every stage of the process, then the remaining unplaced rectangles will be more likely to be placeable. Note that our local waste measure is only an approximate measure of the actual amount of wasted space.

The **maximum fitness number** rule favors placements that result in a "smoother" skyline that contains fewer line segments, which is more likely to allow the placement of larger rectangles without producing wasted space.

Given multiple placements with least waste and highest fitness number, the **earliest in sequence** criterion introduces a tie-breaking rule. This allows us to consider different ways of prioritizing the placement of rectangles using different input sequences.

We provide some comments on the efficient implementation of this heuristic in Section A in the online supplement.

4. Tabu search for 2DRP

For ease of discourse, we will use the notation *2DRP_Heuristic (Seq, H, max_spread)* to represent the use of the heuristic described in the previous section on the rectangle sequence *Seq* on a sheet with dimensions $W \times H$ and the given *max_spread* value. It is contained in a wrapper function *2DRPSolver* that invokes the heuristic several times with different inputs sequences that are generated using a tabu search mechanism, as shown in Algorithm 1.

Algorithm 1: 2DRPSolver (H, iter)

1: **for** each sorting rule **do**
2: $Seq \leftarrow$ sorted sequence of rectangles
3: **for** each maximum spread value *ms* **do**
4: **if** *2DRP_Heuristic (Seq, H, ms)* places all rectangles **then**
5: **return** success
6: **end if**
7: **for** ($i \leftarrow 1; i <$ *iter*; ++i) **do**
8: Generate 10 non-tabu sequences $\{Seq_1, \ldots, Seq_{10}\}$ from *Seq* by swapping two rectangles
9: Let Seq_x be the sequence with highest area utilization using *2DRP_Heuristic* (Seq_x, H, ms)
10: **if** *2DRP_Heuristic* (Seq_x, H, ms) places all rectangles **then**
11: **return** success
12: **end if**
13: Let (b_i, b_j) be the rectangles swapped to produce Seq_x from *Seq*
14: Add (b_i, b_j) to the tabu list for the next $3n$ iterations
15: $Seq \leftarrow Seq_x$
16: **end for**
17: **end for**
18: **end for**
19: **return** failure

We make use of the following six sorting rules to generate the initial input sequence *Seq* for our 2DRP heuristic (line 1):

1. Sort by area in decreasing order
2. Sort by width in decreasing order
3. Sort by height in decreasing order
4. Sort by perimeter in decreasing order
5. Sort by maximum of width and height in decreasing order
6. Sort by length of diagonal + width + height (i.e., perimeter of triangular half) in decreasing order

Let *mh* be the height of the tallest rectangle. For each sequence of rectangles, we try four maximum spread values given by the set $\{mh, mh + (H - mh) * \frac{1}{3}, mh + (H - mh) * \frac{2}{3}, H\}$, as given in line 3.

For each of these combinations of sequences and *max_spread* values, we attempt to optimize the solution in terms of area utilization using a tabu search mechanism for a number of iterations determined by the input parameter *iter* (lines 7–16). The neighborhood operator simply swaps two rectangles in the current sequence. We generate 10 such sequences that are not forbidden by the tabu list, select the one that produces the solution with the highest area utilization, and insert the swap into the tabu list; for the next $3n$ iterations, where n is the number of rectangles in the problem instance, this swap is forbidden when generating sequences.

At any point in the process, if a packing pattern is found that places all n rectangles, then the procedure halts with success. If this does not occur, then *2DRPSolver* reports a failure. In either case, the best packing pattern found in terms of area utilization over the course of the entire algorithm is recorded and returned as the 2DRP solution.

5. Iterative doubling binary search for 2DSP

Our solution for the 2DSP uses the *2DRPSolver* procedure as a subroutine in an iterative doubling binary search as given in Algorithm 2. Let $LB1$ be the naive lower bound computed as the total area of all rectangles divided by the width of the sheet, i.e., $LB1 = \lceil$ total rectangle area/ $W \rceil$. For the fixed orientation variant, we can further strengthen the lower bound. Observe that if there are rectangles with width greater than $W/2$, then these rectangles cannot be placed side by side on the sheet. Hence, the sum of the heights of all such rectangles is also a valid lower bound, which we denote by $LB2$. Also note that if there are rectangles of width exactly $W/2$, then no packing can have a height less than half the total height of these rectangles; we set $LB3$ to be this value. Furthermore, such rectangles cannot be placed side by side with rectangles of width greater than $W/2$ on the sheet. Hence, $LB2 + \lceil LB3 \rceil$ is also a valid lower bound (this is a special case of the bound L_2 used by Martello et al. (2003) where $\alpha = W/2$). We set our initial lower bound LB to be $LB1$ for the rotatable variant, and $LB = \max\{LB1, LB2 + \lceil LB3 \rceil\}$ for the fixed orientation variant.

Several other lower bound measures exist in literature. These can be broadly classified into three categories. Our bounds are based on geometric considerations like (Martello et al., 2003; Bortfeldt, 2006; Iori et al., 2003; Alvarez-Valdes et al., 2009), while other bounds can be derived by relaxing a mathematical model (Scheithauer, 1999; Belov et al., 2008) or using dual feasible functions (Fekete and Schepers, 2004; Boschetti and Montaletti, 2010; Carlier et al., 2007; Clautiaux et al., 2007a,b,c, 2008; Alvarez-Valdes et al., 2009). We refer the reader to Boschetti and Montaletti (2010) for a thorough overview of existing bounds.

We set the upper bound UB to be $LB \times 1.1$. We then perform a binary search on the height of the sheet H. If a feasible solution is found, we record the solution and update the upper bound UB to be the height of the current solution. If a feasible solution cannot be found in the given number of iterations, then proceed with the binary search by setting the lower bound to be $H + 1$. Note that the initial upper bound of $UB = LB \times 1.1$ may be too optimistic, and no solution can be found within the prescribed number of iterations with height no larger than UB. If this is the case, then we increase the upper bound once again by 10% and repeat the process (line 15).

After each binary search attempt, we double the number of iterations of tabu search allowed by the *2DRPSolver* subroutine. Since our lower bound may be far from the optimal value, our binary search must check several values of H, some of which may be infeasible. Given an infeasible value of H, we are unable to quickly determine if it is feasible or infeasible. The rationale behind this iterative doubling approach is based on the observation that our heuristic can never find a feasible solution to infeasible instances, so we wish to minimize the amount of effort spent on infeasible values of H. Early in the process, we spend a small amount of computational effort for each value of H in the binary search to determine a good upper bound by quickly solving "easy" 2DRP instances. Later in the process, when more computational effort is required to find a better feasible solution, we increase the effort by doubling it after each iteration. This iterative doubling technique allows us to apportion a larger amount of computation time to the latter parts of the process to increase the probability of finding good feasible solutions.

Algorithm 2: Iterative Doubling Binary Search for 2DSP

```
1: Compute lower bound LB
2: UB ← LB × 1.1
3: iter ← 1;UBfound ← false
4: while time limit not exceeded and LB ≠ UB do
5:    tempLB ← LB
6:    while tempLB < UB do
7:       H ← ⌊(tempLB + UB)/2⌋
8:       if 2DRPSolver (H,iter) is successful then
9:          Record solution in Sol
10:         UB ← H;UBfound ← true
11:      else
12:         tempLB ← H + 1
13:      end if
14:   end while
15:   if UBfound = = false then UB ← UB × 1.1
16:   iter ← iter∗2
17: end while
18: return Sol
```

6. Computational experiments

Our algorithms were implemented as sequential algorithms in C++ and compiled by GCC 4.1.2, and no multi-threading was explicitly utilized. It was executed on an Intel Xeon E5430 clocked at 2.66 GHz (Quad Core) with 8 GB RAM running the CentOS 5 linux operating system.

6.1. Benchmark test data

In order to evaluate the performance of our algorithms, we used a wide range of benchmark test instances for 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 test sets with known perfect optimal solutions are as follows:

- The 21 instances in test set *C* were generated by Hopper and Turton (2001). They are divided into seven categories, each category containing three instances.
- Hopper (2000) generated 70 instances divided into two sets. The first set *T* corresponds to guillotine patterns, and the second set *N* corresponds to non-guillotine patterns.
- Babu and Babu (1999) proposed a single test instance with 50 input rectangles, which we label as *Babu*.
- The test set *Burke* was generated by Burke et al. (2004) to test their best fit algorithm.
- The *CX* test set generated by Pinto and Oliveira (2005) contains large instances, with the number of input rectangles ranging from 50 to 15,000.
- Wang and Valenzela (2001) provided data instances with known perfect optimal solutions of height 100, but the dimensions of the input rectangles are real numbers. The data is divided into two types: the test set *Nice* involves similar rectangles while the set *Path* use rectangles with vastly differing dimensions. Each data set contains instances ranging from 25 to 1000 rectangles. These instances are made integral by multiplying the original data by 10 and rounding to the nearest integer. Although this invalidates the original perfect optimal solutions, they serve as close approximations and are treated as such.

The remaining test sets were not produced by cutting the sheet into pieces and they do not have known perfect optimal solutions:

- The *ngcut* test set was generated by Beasley (1985b)
- The *gcut* test set was generated by Beasley (1985a)
- The *cgcut* test set was generated by Christofides and Whitlock (1977)
- The *beng* test set was generated by Bengtsson (1982)
- The *bwmv* test set was originally designed for the bin packing problem. It consists of 500 instances divided into 10 classes, each with 5 groups of 10 instances. Six of the classes were created by Berkey and Wang (1987), and the remaining four by Martello and Vigo (1998).
- Bortfeldt and Gehring (2006) generated the *AH* test set, comprising 360 large instances involving 1000 rectangles.

The characteristics of the benchmark test data are summarized in Section B in the online supplement.

6.2. Parameter tuning

There are only two parameters in our tabu search, namely the number of sequences generated in each iteration (*neighborhood size*) and the number of iterations where a swap is forbidden (*tabu tenure*). We followed a common practice to select these values. First, we randomly select a single instance from each test set as a representative sample. We then tried various combinations of values for the neighborhood size and tabu tenure on this sample, and pick the combination that performs best.

Preliminary experiments indicate that a neighborhood size between 5 and 15 and a tabu tenure of between $2n$ and $4n$ (where n is the number of rectangles in the instance) produce the best results. We followed the 2^2 factorial design with center point to determine the best values within these ranges following the practice of design of experiments. For the neighborhood size factor, we tried the two levels $\{5, 15\}$, and for the tabu tenure factor we tried the two levels $\{2n, 4n\}$ with the center located at $(10, 3n)$.

The results of these experiments show that the center point $(10, 3n)$ is the setting that achieves the best performance. Hence, all our subsequent experiments are based on a neighborhood size of 10 and a tabu tenure of $3n$.

6.3. Results for 2DRP

The current best results for the 2DRP are held by the hybrid simulated annealing (HSA) approach by Leung et al. (2012). We compared our *2DRPSolver* (Algorithm 1), which is a simple tabu search using our 2DRP heuristic, with HSA on the test sets used

by the authors in the original publication, which considered the fixed orientation variant only. In our approach, we invoke the *2DRPSolver* procedure several times starting with *iter* = 1, and then double the value of *iter* each time until a time limit of 10 CPU seconds is exceeded, whereupon we return the best result found so far. The time limit given to the HSA approach is 60 CPU seconds, and the results for both sets of experiments were achieved on comparable machine configurations (see Section C in the online supplement).

The results are shown in Table 1. Both algorithms were executed 10 times for each instance. The percentage of unutilized space in the final solutions for the 10 executions was averaged for each instance; the columns labeled $loss_{avg}$ give the average of these values over all instances in the test set. The average amount of time over all executions for the corresponding algorithm to terminate is reported under column t(s). Note that both algorithms will terminate before reaching their respective time limits once a solution with all rectangles loaded is found. When (Leung et al., 2012) reported the results of HSA on the test set *Nice/Path*, they also included an additional 60 test instances obtained from the authors of Wang and Valenzela (2001). Consequently, we also included these additional instances for a fair comparison.

The better result is highlighted in bold. Our approach outperforms HSA for eight out of ten test sets (except *cgcut* and *gcut*). These results indicate that our heuristic combined with a simple tabu search can be considered the best existing 2DRP algorithm over these benchmark test cases.

An inspection of the *cgcut*, *gcut* and *ngcut* test sets reveal that these sets contain several "large" rectangles of width greater than $W/2$, where W is the width of the sheet, so no two such large rectangles can be placed side by side on the sheet. Our approach tends to perform relatively poorly for these types of instances. Consider the example in Fig. 6(a), where the two large rectangles a and b have already been placed, and rectangles c and d are the only remaining rectangles that can fit into the space beside a and b. Our heuristic may first place c on top of b (generating no local waste), and then the *only fit* rule would place d on segment s_3, resulting in a large amount of wasted space. Instead, placing both c and d in the area beside a and b is optimal (e.g., Fig. 6(b)). Note that this issue mainly arises in the fixed orientation variant.

Instances with several large rectangles may not be common in practice. Also, we can partially address this issue by reducing such instances to equivalent problems with fewer large rectangles using the method proposed by Boschetti and Montaletti (2010). Further refinements of our heuristic to handle this special case is a possible direction for future work.

6.4. Results for 2DSP

We first analyze the results for the rotatable variant, which allows the rectangles to be rotated by 90°. We compare our iterative doubling binary search (IDBS) approach with the best approaches for the 2DSP in existing literature:

- **BF**: a deterministic placement heuristic by Burke et al. (2004) based on the best fit strategy.
- **BF + SA**: a simulated annealing enhancement of the BF heuristic by Burke et al. (2009).
- **BBF**: a bi-directional best-fit heuristic by Aşık and Özcan (2009).
- **FH**: an iterative greedy heuristic by Leung and Zhang (2010).

The computational environments for these approaches are summarized in Section C in the online supplement.

In the existing work on the rotatable variant of 2DSP, the test sets used all had known perfect optimal solutions. Consequently, we also evaluate our IDBS approach on these sets of instances; the results are given in Table 2. For each test instance, the *relative gap* of the height of the produced solution *sol* from the optimal solution *opt* is computed as $(sol - opt)/opt$. The original authors for BF, FH and BBF heuristics only executed their approaches once for each instance; we report the average relative gaps for each set of instances in the columns *rg*. The original authors executed the BF + SA method 10 times for each instance, and each execution was given a time limit of 60 CPU seconds. The values in the column rg_{avg} is computed by first finding the average relative gap over the 10 executions for each instance, and then taking the average over the test set. Note that the amount of time required to execute the BF heuristic is negligible. For FH and BBF, the columns $t(s)$ report the average time taken over all instances in the test set. A dash ('–') indicates that the result was not reported in the corresponding publication.

For each instance, we ran our IDBS algorithm 10 times with a time limit of 100 CPU seconds. The column rg_{best} gives the relative gap from optimal of the best solution found out of the 10 executions on each instance, averaged over the number of instances. Similarly, rg_{worst} reports the value for the worst soluions out of the 10 executions on each instance. Finally, the column $t^*(s)$ gives the average time required for IDBS to first find its final best solution for the test set.

The results show that IDBS outperforms all existing heuristics for the rotatable variant of the 2DSP. For the *C*, *Babu* and *Burke* test sets, IDBS is able to find the optimal solution in every execution. For the remaining test sets, the average height of the solutions found by IDBS is at most 1.31% away from optimal, which is the best result in literature. Furthermore, even when the worst solutions from 10 executions for each instance are taken, the average gap to optimal is at most 1.43%. The detailed results for each test set can be found in Section D in the online supplement.

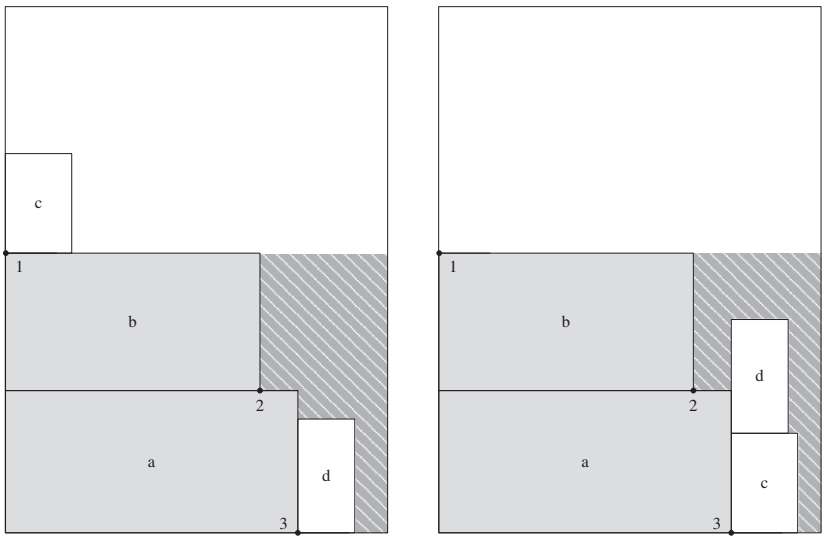
The columns under the heading *No Tabu* give the results of our approach when we remove the tabu search component (lines 7–16 in Algorithm 1), which reduces our algorithm to a simple binary search on the height H of the packing. For each height value, this algorithm considers 4 different spread values for each of the 6 initial input sequences, for a total of 24 solutions. We see that aside from the single *Babu* class instance, this algorithm performs significantly worse than IDBS for all test sets, which shows the impact of our tabu search to find alternative input sequences.

For the fixed orientation variant, the GRASP approach by Alvarez-Valdes et al. (2008) is the current best algorithm. Hence, we only compare our IDBS algorithm with this GRASP approach. Both groups of test instances (with and without known perfect optimal solutions) were employed. The results are summarized in Table 3.

The GRASP approach was executed 10 times for each instance and was given a time limit of 60 CPU seconds per execution. We set the time limit for our IDBS approach at 100 CPU seconds for the test sets with known perfect optimal solutions, and 200 CPU seconds for the other sets. All entries in the table are averaged over

Table 1
Results for 2DRP.

Test set	HSA		*2DRPSolver*	
	$loss_{avg}$	t(s)	$loss_{avg}$	t(s)
C	0.19	41.27	**0.06**	4.90
N	1.62	53.59	**1.16**	6.73
T	1.62	52.79	**1.16**	6.67
Burke	0.08	33.06	**0.00**	2.17
CX	0.36	26.50	**0.33**	4.68
cgcut	**3.47**	40.07	3.97	7.22
gcut	**10.43**	60.02	11.12	7.87
ngcut	14.86	25.02	**14.02**	3.64
beng	0.67	6.05	**0.57**	0.07
Nice/Path	0.65	60.05	**0.61**	10.75



(a) Our heuristic (b) Optimal solution

Fig. 6. Possible poor performance due to large rectangles.

Table 2
Results for rotatable variant.

Test set	BF	BF + SA	FH		BBF		IDBS				No tabu	
	rg	rg_{avg}	rg	t(s)	rg	t(s)	rg_{avg}	rg_{best}	rg_{worst}	t^*(s)	rg_{avg}	t(s)
C	6.10	2.33	1.71	2.09	2.77	0.3	**0**	**0**	**0**	0.31	1.06	0.02
Babu	6.67	6.67	–	–	6.67	0.4	**0**	**0**	**0**	0.02	0.00	0.00
Burke	4.05	1.78	1.41	0.78	2.24	13	**0**	**0**	**0**	0.41	0.81	0.04
N	–	–	–	–	–	–	0.93	0.80	1.03	19.11	4.23	0.04
T	–	–	–	–	3.90	211	**1.01**	**0.80**	**1.20**	19.48	4.09	0.05
CX	–	–	1.02	0.87	2.52	922	**0.22**	**0.19**	**0.26**	13.51	1.33	0.41
Nice	5.85	4.19	–	–	5.75	9.80	**1.06**	**0.92**	**1.27**	44.93	2.87	1.73
Path	6.88	3.13	–	–	5.95	8.8	**1.31**	**1.17**	**1.43**	48.87	2.72	3.61

Table 3
Results for fixed orientation variant.

Test set	GRASP		IDBS				No Tabu	
	rg_{avg}	rg_{best}	rg_{avg}	rg_{best}	rg_{worst}	t^*(s)	rg_{avg}	t(s)
C	0.98	0.85	**0.14**	**0.04**	**0.20**	10.46	1.34	0.03
Babu	0.00	0.00	0.00	0.00	0.00	0.04	0.00	0.00
Burke	0.91	0.91	**0.00**	**0.00**	**0.00**	4.73	1.73	0.32
N	2.41	2.31	**1.29**	**0.87**	**1.94**	28.38	5.27	0.05
T	2.27	2.16	**1.33**	**1.04**	**1.79**	21.55	5.67	0.05
CX	1.02	0.98	**0.43**	**0.40**	**0.45**	17.96	1.93	0.70
Nice	3.55	3.33	**1.96**	**1.80**	**2.12**	42.54	3.07	1.14
Path	2.75	2.61	**2.03**	**1.91**	**2.15**	48.72	3.68	2.58
beng	0.00	0.00	0.00	0.00	0.00	0.04	0.51	0.00
cgcut	**2.37**	**2.37**	2.42	2.42	2.42	37.18	4.60	0.01
gcut	**5.43**	**5.42**	5.63	5.55	5.71	56.76	9.02	0.04
ngcut	**1.34**	**1.34**	1.60	1.60	1.60	0.19	3.58	0.00
bwmv	–	–	3.01	2.86	3.13	39.11	5.43	0.06
bwmv*	**1.80**	1.73	1.86	**1.71**	1.98	39.11	4.20	0.06
AH	–	–	0.87	0.77	0.95	104.06	1.36	13.06

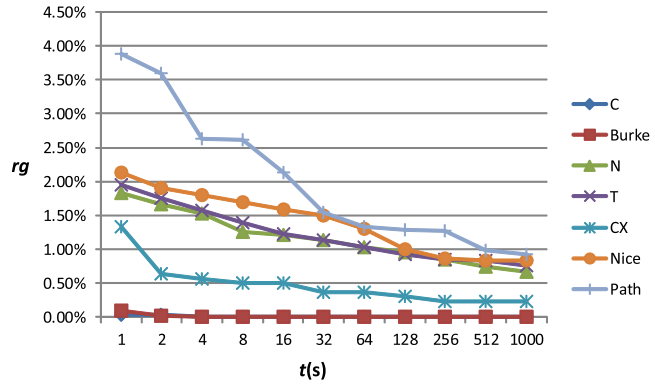
all instances in the test sets; the columns have the same meanings as in Table 2. The best results are shown in bold.

For the instances with known perfect optimal solutions, IDBS is superior to GRASP. For many test sets, the average relative gap from optimal achieved by GRASP is about twice that of IDBS. In fact, for every test set, the average gap to optimal of the worst solutions found from 10 executions of each instance for IDBS is equal or superior to the average gap to optimal of the best solutions found from 10 executions of each instance for GRASP. Note that IDBS was able to find the optimal solution for the *Babu* instance and *Burke* test set within 5 seconds.

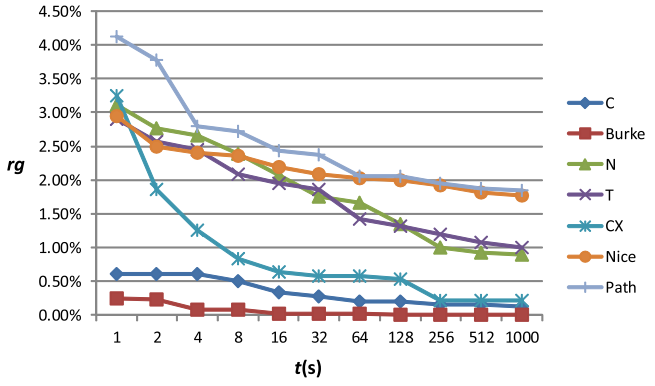
For the remaining instances, since the optimal solutions are unknown, the relative gap values are computed in terms of the naive lower bound $LB1$. There are some where IDBS produces a better solution than GRASP and others where GRASP is superior, and both approaches found the optimal solutions for all instances in the *beng* test set. On average, GRASP outperforms IDBS by a very small amount (less than 0.3%) for these instances. Note that these instances contain several large rectangles, which tend to cause our heuristic to perform relatively poorly (as discussed in Section 6.3). Overall, IDBS outperforms GRASP for the entire set of test instances considered.

Finally, the columns under the heading *No Tabu* verify that the tabu search component of IDBS is similarly signifcant for the fixed orientation variant.

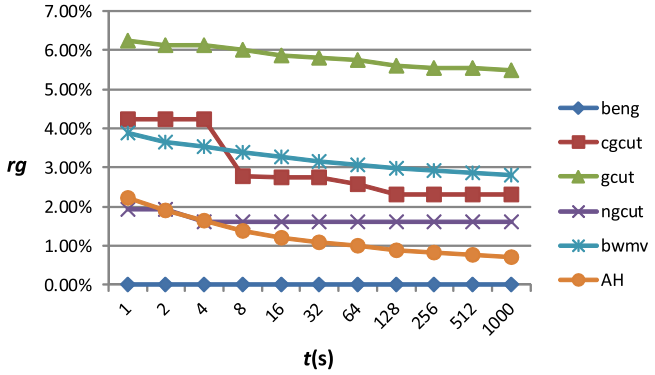
We report the results for the *bwmv* test set twice. The first set of values labeled *bwmv* is calculated in the same way as the other entries in the table. The values for the test set labeled *bwmv** were calculated according to the method used by Alvarez-Valdes et al. (2008). Recall that the *bwmv* test set is divided into 10 classes, each class containing 5 groups. For each group, (Alvarez-Valdes et al., 2008) calculated the group average as follows: take the total height of all solutions for each instance averaged over 10 executions, sum them up, and find the relative gap between this value and the total of the lower bounds for all instances in the group, i.e., $(\sum_{i\in group} H_i)/(\sum_{i\in group} LB_i) - 1$ where H_i is the average height of the



(a) Rotatable variant



(b) Fixed variant, perfect optimal



(c) Fixed variant, unknown optimal

Fig. 7. IDBS convergence.

solutions over 10 executions for instance i in the group. The class average is the average of the group averages, and the reported value is the average of the class averages over the 10 classes. This method of computing group average is non-standard, since instances with a large lower bound may have a greater effect on this value than instances with a small lower bound. Furthermore, the method used by the authors to compute their lower bounds LB_i is different from ours. Therefore, our values for $bwmv^*$ were computed using their reported lower bound values.

The detailed results for all test instances for the fixed orientation variant are located in Section D in the online supplement.

The graphs in Fig. 7 give an indication of the convergence behavior of our IDBS approach. For each test instance, we executed IDBS for 1000 CPU seconds and plotted the average relative gap achieved by IDBS for each test set as the computation time increases; note that the x-axis values increase exponentially. As expected, the convergence behavior varies depending on the test set, but for most test sets the solution found after 8 CPU seconds is not much worse that the solution found after 1000 CPU seconds. Hence, our IDBS approach exhibits rapid convergence for most of these test sets. A visual inspection of Fig. 7(a) and (b) suggests that a computation time limit of about 100 CPU seconds is sufficient to achieve a high-quality solution for the test sets with known perfect optimal solutions, and Fig. 7(c) suggests a computation time limit of about 200 CPU seconds for the remaining test sets. Although improvements continue when more time beyond these limits is spent in most cases, they are achieved at a diminishing rate.

7. Conclusions

In this paper, we present a greedy heuristic for solving the 2DRP involving several priority rules that are motivated by observations on the nature of 2D packing problems. A tabu search procedure that employs this 2DRP heuristic outperforms the best existing approach for this problem on benchmark test instances. We then use this tabu search procedure as a subroutine in an iterative doubling binary search (IDBS) on the height of the sheet to solve the 2DSP. The resultant approach outperforms all existing approaches for the 2DSP on both the fixed and rotatable variants on a large set of benchmark test cases.

Our greedy 2DRP heuristic is based on the skyline representation of packing patterns. While this representation has the advantage of efficient computation, it also eliminates some positions that can contain rectangles. For example, in Fig. 4(a) there may be rectangles that can be placed on segment s_4 or s_5, but these positions are not preserved by the skyline representation. It may be possible to produce a better heuristic using a more complete representation of packing patterns, taking into account a possible tradeoff in terms of computational efficiency.

To the best of our knowledge, the use of iterative doubling in a binary search as realized in our IDBS approach has never been previously proposed. This idea may also be applicable to binary search algorithms on other problems to improve efficiency.

Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, at doi:10.1016/j.ejor.2011.06.022. Detailed solution files can be downloaded from www.computational-logistics.org/orlib/2dsp/index.html.

References

Alvarez-Valdes, R., Parreño, F., Tamarit, J.M., 2008. Reactive grasp for the strip-packing problem. Computers & Operations Research 35, 1065–1083.
Alvarez-Valdes, R., Parreño, F., Tamarit, J., 2009. A branch and bound algorithm for the strip packing problem. OR Spectrum 31, 431–459.
Aşık, O., Özcan, E., 2009. Bidirectional best-fit heuristic for orthogonal rectangular strip packing. Annals of Operations Research 172, 405–427.
Babu, A.R., Babu, N.R., 1999. Effective nesting of rectangular parts in multiple rectangular sheets using genetic and heuristic algorithms. International Journal of Production Research 37, 1625–1643.
Baker, B.S., Coffman, E.G., Rivest, R.L., 1980. Orthogonal packings in two dimensions. SIAM Journal on Computing 9, 846–855.
Beasley, J.E., 1985a. Algorithms for unconstrained two-dimensional guillotine cutting. The Journal of the Operational Research Society 36, 297–306.
Beasley, J.E., 1985b. An exact two-dimensional non-guillotine cutting tree search procedure. Operations Research 33, 49–64.
Belov, G., Scheithauer, G., Mukhacheva, E.A., 2008. One-dimensional heuristics adapted for two-dimensional rectangular strip packing. Journal of the Operational Research Society 59, 823–832.
Bengtsson, B.E., 1982. Packing rectangular piecesa heuristic approach. The Computer Journal 25, 253–257.
Berkey, J.O., Wang, P.Y., 1987. Two-dimensional finite bin-packing algorithms. The Journal of the Operational Research Society 38, 423–429.

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

Bortfeldt, A., Gehring, H., 2006. New large benchmark instances for the two-dimensional strip packing problem with rectangular pieces. In: Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06). IEEE, p. p. 30b.

Boschetti, M.A., Montaletti, L., 2010. An exact algorithm for the two-dimensional strip-packing problem. Operations Research 58, 1774–1791.

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

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

Carlier, J., Clautiaux, F., Moukrim, A., 2007. New reduction procedures and lower bounds for the two-dimensional bin packing problem with fixed orientation. Computers & Operations Research 34, 2223–2250.

Chazelle, 1983. The bottom-left bin-packing heuristic: an efficient implementation. IEEE Transactions on Computers C-32, 697–707.

Christofides, N., Whitlock, C., 1977. An algorithm for two-dimensional cutting problems. Operations Research 25, 30–44.

Clautiaux, F., Carlier, J., Moukrim, A., 2007a. A new exact method for the two-dimensional bin-packing problem with fixed orientation. Operations Research Letters 35, 357–364.

Clautiaux, F., Carlier, J., Moukrim, A., 2007b. A new exact method for the two-dimensional orthogonal packing problem. European Journal of Operational Research 183, 1196–1211.

Clautiaux, F., Jouglet, A., Elhayek, J., 2007c. A new lower bound for the non-oriented two-dimensional bin-packing problem. Operations Research Letters 35, 365–373.

Clautiaux, F., Jouglet, A., Carlier, J., Moukrim, A., 2008. A new constraint programming approach for the orthogonal packing problem. Computers & Operations Research 35, 944–959.

Dowsland, K., 1993. Some experiments with simulated annealing techniques for packing problems. European Journal of Operational Research 68, 389–399.

Fekete, S.P., Schepers, J., 2004. A general framework for bounds for higher-dimensional orthogonal packing problems. Mathematical Methods of Operations Research 60, 311–329.

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

Hifi, M., 1998. Exact algorithms for the guillotine strip cutting/packing problem. Computers & Operations Research 25, 925–940.

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

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

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

Imahori, S., Yagiura, M., 2010. The best-fit heuristic for the rectangular strip packing problem: An efficient implementation and the worst-case approximation ratio. Computers & Operations Research 37, 325–333.

Iori, M., Martello, S., Monaci, M., 2003. Metaheuristic algorithms for the strip packing problem. In: Pardalos, P.M., Korotkikh, V. (Eds.), Optimization and Industry: New Frontiers, Applied Optimization, vol. 78. Springer, pp. 159–179 (Chapter 7).

Kenmochi, M., Imamichi, T., Nonobe, K., Yagiura, M., Nagamochi, H., 2009. Exact algorithms for the two-dimensional strip packing problem with and without rotations. European Journal of Operational Research 198, 73–83.

Lesh, N., Marks, J., McMahon, A., Mitzenmacher, M., 2004. Exhaustive approaches to 2d rectangular perfect packings. Information Processing Letters 90, 7–14.

Leung, S.C.H., Zhang, D., 2010. A new heuristic approach for the stock-cutting problems. In: Engineering and Technology K: Business and Economic Sciences 2:2. World Academy of Science, pp. 121–126.

Leung, S.C.H., Zhang, D., Zhou, C., Wu, T., 2012. A hybrid simulated annealing metaheuristic algorithm for the two-dimensional knapsack packing problem. Computers & Operations Research 39, 64–73 [Special Issue on Knapsack Problems and Applications.].

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

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

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

Oliveira, J.E., Ferreira, J.S., 1993. Algorithms for nesting problems. In: Applied simulated annealing. LNEMS, vol. 396. Springer, pp. 255–274.

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

Scheithauer, G., 1999. Lp-based bounds for the container and multi-container loading problem. International Transactions in Operational Research 6, 199–213.

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

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

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

Wu, Y.-L., Huang, W., Lau, S.-c., Wong, C.K., Young, G.H., 2002. An effective quasi-human based heuristic for solving the rectangle packing problem. European Journal of Operational Research 141, 341–358.

Zhang, D., Liu, Y., Chen, S., Xie, X., 2005. A meta-heuristic algorithm for the strip rectangular packing problem. In: Wang, L., Chen, K., Ong (Eds.), Advances in Natural Computation chapter 157, Lecture Notes in Computer Science, vol. 3612. Springer Berlin/Heidelberg, Berlin, Heidelberg, pp. 1235–1241.

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