A fast layer-based heuristic for non-guillotine strip packing

Stephen C.H. Leung\textsuperscript{a}, Defu Zhang\textsuperscript{b,}\textsuperscript{*}

\textsuperscript{a}Department of Management Sciences, City University of Hong Kong, Hong Kong
\textsuperscript{b}Department of Computer Science, Xiamen University, Xiamen 361005, China

**Abstract**

In this paper, an orthogonal strip packing problem with rotation of items and without the guillotine packing constraint is considered. A fast heuristic algorithm for the large-scale problems is presented. This heuristic algorithm is mainly based on heuristic strategies inspired by the wall-building rule of bricklayers in daily life. The heuristics is simple and the setting of parameter is not required. Each layer is initialized with either a single item or a bunch of equal-width items. The remaining part of the layer is filled by a bottom-left strategy preferring items which eliminate corners of the current layout. Items can also be placed across several layers. Then, the evaluation rule, which is based on the fitness value for different rectangles to a given position, is able to select an appropriate rectangle to pack. The computational results on a broad range of benchmark problems show that the fast layer-based heuristic algorithm can compete with other latest heuristics and meta-heuristics from the literature in terms of both solution quality and computational time. The fast layer-based heuristic algorithm can compete with the latest published algorithms. In particular, it performs better for large-scale problem instances.

© 2011 Elsevier Ltd. All rights reserved.

---

**1. Introduction**

**1.1. Problem statement**

Cutting and packing problems belong to a well-known family of combinatorial optimization problems and have many industrial applications in the different fields of operations research (Burke, Kendall, & Whitwell, 2004). For example, in the wood or glass industries, it is necessary to consider how to cut rectangular pieces from large sheets of material. In the warehousing field, it is needed to study how to place goods on shelves. In the newspapers typesetting field, it is important to arrange articles and advertisements in pages. In the shipping industry, it is required to investigate how to ship a set of objects of various sizes as many as possible in a larger container. In the optic-fiber communication field, it is essential to understand how to accommodate a bunch of optical fibers in a pipe as small as possible. In floor planning, it is necessary to consider the very-large-scale integration (VLSI) design. These applications can be formalized as a cutting and packing problem with different constraints and objectives (Lodi, Martello, & Monaci, 2002b, 2002a). One of the goals in most industrial applications is to produce a good quality of arrangements of items on the stock sheet in order to maximize material utilization or minimize wastage. Moreover, there is a goal which is to produce a solution within a reasonably short computational time. The latter goal is especially important in the logistic fields because any delay will lead to a loss of customers. In some industrial fields, the cutting and packing task is always performed by skilled workers. However, due to a lack of material and the need of industrial applications, automated-packaging algorithms have become more widely used in past years (Li & Milenkovic, 1995). For more algorithms or reviews on cutting and packing problems, the interested reader is referred to the literature (Dowsland & Dowsland, 1992; Lodi et al., 2002a, 2002b; Oliveira José & Wäscher, 2007).

In this paper, the two-dimensional orthogonal strip packing problem is considered without any guillotine constraint, for instance, cutting glass parts in the glass industry. This problem can be stated as follows: Given a rectangular sheet of a width and an infinite height and a set of rectangles with arbitrary sizes, the orthogonal strip packing problem is to place each rectangle on the sheet so that no two rectangles overlap and the height \( h \) of the used sheet is minimized. Let \( W \) be the width of the rectangular sheet, and \( n \) is the number of rectangles (or items). Let \( h_i \) and \( w_i \) be the height and width of rectangle \( i \) (\( 1 \leq i \leq n \)), respectively. A more detail and formal statement for this problem is given in Zhang, Kang, and Deng (2006). In this paper, we assume that the edges of each rectangle are parallel to the edges of the rectangular sheet (orthogonal packing). In addition, all rectangles must be placed into the rectangular sheet and are allowed to rotate 90°. Bortfeldt (2006) defined this problem as RF subtype and Wäscher, Häußner, and Schumann (2007) classified the problem as a two-dimensional single large object placement problem.
1.2. Literature review

The two-dimensional orthogonal strip packing problem, which belongs to a subset of classical cutting and packing problems, has been shown to be NP hard (Beasley, 1985; Lodi et al., 2002a, 2002b). Some optimal algorithms for orthogonal two-dimension cutting and packing problem were proposed in Beasley (1985), Christofides and Hadjiconstantinou (1995) and Hifi and Zissimopoulos (1997). However, they might not be practical for large-scale problems because of the large amount of computational time required to obtain a desirable solution. Therefore, heuristic algorithms, which can produce good approximation solutions in an acceptable computational time, for example, the well-known bottom-left (BL), bottom-left-fill (BLF) and other heuristic methods (Baker, Coffman, & Rivest, 1980; Berkey & Wang, 1987; Chazelle, 1983), are preferred to solving this class of problems. These heuristic algorithms can produce good solutions for some problems in a reasonably short computational time. Some hybrid algorithms combining genetic algorithm (GA) with deterministic heuristic methods for the orthogonal packing problem have been proposed, for example, BL and GA (Jakobs, 1996; Liu & Teng, 1999; Ramesh Babu & Ramesh Babu, 1999), neural network and GA (Dagli & Poshyanonda, 1997); A simulated annealing (SA) algorithm (Lai & Chan, 1996) and other algorithms (Christofides & Whitlock, 1997) are presented to solve the cutting stock problem. An empirical investigation of meta-heuristic and heuristic algorithms of the orthogonal packing problem of rectangles is given by Hopper and Turton (2001). However, generally speaking, those non-deterministic algorithms are still time-consuming and less practical for problems with a large number of rectangles. Recently, several exact approaches (ClaUTiaux, Carlier, & Moukrin, 2007; Fekete, Schepers, & van der Veen, 2006; Hifi & M’Hallah, 2005; Martello, Monaci, & Vigo, 2003) have been presented to solve this class of problems. However, they are still confined to small scale problems. Promisingly, some new heuristic algorithms, for example, quasi-human heuristic (Wu, Huang, Lau, Wong, & Young Gilbert, 2002), constructive approach (Hifi & M’Hallah, 2003; Hifi & M’Hallah, 2006), new fast placement best-fit (BF) heuristic (Burke et al., 2004), heuristic recursion algorithm (Zhang et al., 2006), hybrid meta-heuristic algorithms (Bortfeldt, 2006; Iori, Martello, & Monaci, 2003; Dowsland Kathryn, Herbert Edward, Kendall, & Burke, 2006; Alvarez-Valdes, Parreño, & Tamarit, 2007; Gonzáles José, 2007; Hadjiconstantinou & Iori, 2007; Burke Edmund, Kendall, and Whitwell (2009)) and cutting-plane approach (Baldacci & Boscetti Marco, 2007) have been developed to solve this class of packing problems. These heuristics are fast and effective. In particular, the best-fit (BF) heuristic developed by Burke et al. (2004) cannot only be very efficient, but also find better solutions than some well-known meta-heuristics. BF + metaheuristic is further developed by adding a metaheuristic phase that first uses BF to pack some rectangles and then applies BLF + metaheuristic for the remaining rectangles (Burke Edmund et al. (2009)). A genetic algorithm called SPGL which works directly on the solution layouts and does not require any encoding of the solutions is presented by Bortfeldt (2006). A very effective heuristic algorithm called heuristic rectangle packing (HRP) algorithm, is presented by making use of the corner-occupying action and caving degree strategy to guide the packing (Huang, Chen, & Xu, 2007). The caving degree can reflect the nearness between the rectangle to be packed and its nearest rectangle, so it performs well for the problems with permitting rotation. A new heuristic recursive algorithm combined with branch-and-bound techniques, which is presented by Cui, Yang, Cheng, and Song (2008), is used to solve the rectangular guillotine strip packing problem. A greedy randomized adaptive search procedure (GRASP) is proposed by learning some data to determine the desirable parameter settings for the strip packing problem. GRASP performs better for small instances (Alvarez-Valdes, Parreño, & Tamarit, 2008). Belov, Scheithauer, and Mukhacheva (2008) presented two iterative heuristics based on one-dimensional heuristics, which are SVC(SubKP) (stands for Sequential value correction(substitution knapsack problem)) and BS(BLR) (stands for Bubble search(bottom-left-right)). It is found that SVC is a powerful algorithm and performs well for most instances. Recently, a least waste first heuristic (LWF) is presented by evaluating positions, and is improved by combining simulated annealing algorithm (Wei, Zhang, & Chen, 2009). LWF performs better for rectangle packing problem. SPGAL, GRASP, BF + SA, SVC and LWF are based on heuristic strategies and can find very excellent solutions within a reasonable time, but they are more complex, especially the performance of these algorithms significantly depends on the settings of the parameters. Moreover, some algorithms generally require much time to obtain a desirable solution when the problem scale is larger. HRP does not involve the settings of the parameters, but it requires much more time to find a good solution for large-scale problem instances (Huang et al., 2007). Recently, a new and improved level heuristics is presented. It can quickly find a solution, but the quality of solution is worth improving further. In fact, the efficiency and robustness of the algorithm is very important in many real-world applications, for example, in the logistics field.

1.3. Our contributions

In this paper, a fast layer-based heuristic algorithm for solving the large-scale orthogonal strip packing problems is presented. Computational results on a large amount of benchmark problem instances have shown that the presented heuristic can find better solutions within 60 s and compete with the excellent algorithms, especially, the presented algorithm performs better for considered large-scale instances. The proposed heuristic is simple and no parameter tuning is required. Each layer is initialized with either a single item or a set of equal-width items. The remaining part of the layer is filled by a bottom-left (BL) strategy preferring items which eliminate corners of the current layout. Items can also be placed across several layers.

The main differences between bricklaying heuristic algorithm (Zhang, Han, & Ye 2008) and the proposed fast layer-based heuristic (FH) is that FH introduces the strategy of stacking rectangles, in particular, which rectangle to be selected to pack in FH is determined by the fitness value rule. FH first places one reference rectangle and stacks some rectangles to obtain the reference line. It then finds one lowest available rectangle space under the reference line, computes the fitness value of each unplaced rectangles, and selects one rectangle with the maximum fitness value to place. At last, greedy search is used to improve the placing result.

FH is different from other layer-based algorithms, for example, SC and SCR presented by Ortmann, Ntene, and van Vuuren (2010). They stack downwards onto ceiling-packed items. The entire list of items is sequentially searched to determine whether the items fit onto the floor based on the wasted space. However, for FH, which rectangle to be selected to pack is determined by the size of fitness value.

The rest of this paper is organized as follows. In Section 2, inspired by the wall-building rule of bricklayers in daily life, we present a fast layer-based heuristic algorithm. Computational results are described in Section 3. Conclusions are summarized in Section 4.

2. A fast layer-based heuristic algorithm

A bricklaying heuristic strategy inspired from the work of wall building based on layer is first introduced by Zhang et al. (2008). In this section, the main contribution of this strategy is reviewed.
Then, we extend the idea to develop a fast layer-based heuristics for the considered strip packing problem.

2.1. Motivation

The idea of bricklaying heuristic algorithm is to place rectangles by layer (Zhang et al., 2008). A new layer determined by a reference rectangle starts when the current layer cannot place more rectangles. The layers are divided by the reference line. For example, in Fig 1(a), suppose that the sequence of the rectangles placing $X$ is $\{1, 4, 2, 3\}$, the rectangle 1 is searched as the reference block and is placed into the position O and forms the layer 1. Since rectangle 4 cannot be placed under layer 1, rectangles 2 and 3 are considered to be placed under layer 1 alternatively. In Fig 1(b), a new layer 2 determined by the reference rectangle 4 starts when no rectangles can be placed into the space $S$ under the reference line BC; a new reference line DE is determined by the rectangle 4. The advantage of bricklaying heuristic algorithm is that it can better utilize the space, for example, the space under the reference line BC can be used when placing layer 2. In real life, the bricklayers have accumulated a large number of experiences during the process of building the wall. These experiences include that the wall is built by layer, and that the lowest positions are given a priority to place. Inspired by their experiences, the procedure of fast heuristics for packing problem is as follows:

1. Place one reference rectangle, and form the current layer and the current reference line.
2. From the lowest position to the highest position and from left to right, place the remaining rectangles into the available positions under the current reference line until no rectangles can be placed.
3. If all rectangles are placed, then stop, otherwise go to step 1.

2.2. Layer-based heuristic algorithm

As the reference rectangle determines the size of the space under the reference line, if the height of the current layer is too small, it will easily lead to placing the small rectangles first. Generally, in order to place large rectangles firstly, it is necessary to increase the height of the layer by stacking the rectangles of the same width ($w_i$) or length ($l_i$) as the reference rectangle's width. For example, some small rectangles have to be placed under the reference line BC defined by the rectangle 1 (reference rectangle) in Fig. 2(a). From Fig. 2(b), the height of the layer will be higher if the rectangles 4 and 5 are stacked on the top of the rectangle 1. This allows the large rectangles to be placed first. Originally, the rectangles 2 and 6 cannot be placed under the reference line BC in Fig. 2(a). Under the proposed strategy, they can be placed under the reference line BC in Fig. 2(b). Therefore, in step 1, we suggest to stack rectangles of the same width or length as the reference rectangle's width until the total height of this layer exceeds the lower bound of solution $LB$, where $LB = \sum_{i=1}^{n} l_i w_i / W$. It is noted that the layer may only contain the reference rectangle.
The first case is $h_1 > h_2$. For this case, the unplaced rectangles should be placed close to $h_1$. There are four kinds of possible available placements defined by different rectangle $R$. Which kind of placements is better? The bricklayers have rich experience and know how to place a rectangle by some priority rules. It is observed that one placement is good if it can decrease the number of corner positions. So we present the conception of fitness value to evaluate whether one placement is good or not. If one placement can fit more corner positions, the corresponding rectangle for this placement is given a larger fitness value. In detail, the fitness value of one rectangle $R$ for the first case (see Fig. 3(1)) is given as follows:

1. For Fig. 3(1.1), the placement can fit three corner points, so the fitness value of $R$ is 3. Namely, if rectangle $R$ is placed into the position $p$ and obtains the placement in Fig. 3(1.1), then the fitness value of $R$ is 3. It is a good placement because it can fit three corner points.

2. For Fig. 3(1.2), the placement can fit two corner points, so the fitness value of $R$ is 2. Namely, if rectangle $R$ is placed into the position $p$ and obtains the placement in Fig. 3(1.2), then the fitness value of $R$ is 2. It is noted that the top dotted edge of the rectangle can move vertically, meaning that the rectangles have the same fitness value only if these rectangles can fit the bottom edge of the rectangle space $S$.

3. For Fig. 3(1.3), the placement can fit one corner points, so the fitness value of $R$ is 1. Namely, if rectangle $R$ is placed into the position $p$ and obtains the placement in Fig. 3(1.3), then the fitness value of $R$ is 1. It is noted that the right dotted edge of the rectangle can move horizontally, meaning that the rectangles have the same fitness value only if these rectangles do not touch the right edge of $S$. This kind of placement does not fit the corner position in $p$; it can be recognized that the corner position moves along the arrow to the right (see arrow in Fig. 3(1.3)).

4. For Fig. 3(1.4), the placement cannot fit any corner points, so the fitness value of $R$ is 0. Namely, if rectangle $R$ is placed into the position $p$ and obtains the placement in Fig. 3(1.4), then the fitness value of $R$ is 0. It is noted that the right dotted edge of the rectangle can move horizontally and the top dotted edge of the rectangle can move vertically. It means that the rectangles have the same fitness value only if these rectangles do not touch the right edge of $S$. Similarly, this kind of placement does not fit the corner position in $p$. It can be recognized that the corner position moves along the arrow to the right (see arrow in Fig. 3(1.4)).

5. The fitness value of the rectangle is $-\infty$ if that rectangle cannot be placed into the rectangle space $S$.

For the second case of $h_2 > h_1$, the unplaced rectangles should be placed close to $h_2$. Similarly, there are four kinds of possible avail-
able placements. For each placement in Fig. 3(2.1), (2.2), (2.3) and (2.4), the fitness value of the corresponding rectangle is 3, 2, 1 and 0, respectively. The fitness value of the rectangle is -∞ if that rectangle cannot be placed into the rectangle space $S$. It is noted that the proposed heuristic algorithm is inspired from the process of wall-building, but it has been greatly improved by adding the strategies of stacking brick and fitness value. In particular, the stacking strategy may be different from practical wall-building processes.

For a given position $p$, we can compute the fitness value of all the unplaced rectangles, then select one rectangle with the maximum fitness value to place there. The rectangle in the front of the ordering rectangle sequence is selected to be placed if several rectangles have the same maximum fitness value. In addition, it is very important to determine and place the reference rectangle since other rectangles must refer to the reference line. Due to the long edge placed along the reference line can stack more rectangles, so the long edge of the reference rectangle is placed along the reference line unless the length of long edge is greater than $W$.

In detail, the heuristic algorithm based on the above heuristic strategies is stated in Fig. 4.

It is noted that a new rectangle space will have to be searched if the length or width of the current rectangle space determined by $p$ is less than the minimum length or width of the unplaced rectangles.

2.3. Improvement on layer-based heuristic algorithm

Since the performance of heuristic algorithm significantly depends on the placing ordering $X$ of the rectangles, where ordering $X$ is a sequence of the rectangles placing, some research results have shown that the placing ordering of the rectangles affects the performance of the presented algorithm (Bortfeldt, 2006; Hopper & Turton, 2001; Zhang et al., 2006). In this paper, we develop a heuristic strategy for selecting an initial placing ordering, in which unplaced rectangles should be sorted by a non-increasing ordering of perimeter size before placing. According to this placing ordering, the rectangle with the maximum perimeter is given a higher priority of being placed. The benchmark instances on the two-dimensional orthogonal strip packing problem include 21 problem instances (data set CX: 50cx ranging from 16 to 197 items in Hopper and Turton (2001) and 13 problem instances (data set N: N1 ~ N13) randomly generated by Burke et al. (2004), and especially a large instance involving 3152 items is given. The seven extra large-scale instances (data set CX: 50cx ~ 15000cx) proposed by Pinto and Oliveira (2005) are included. The problem scale of these instances varies from 50 to 15,000 rectangles. The optimal solution of the above 41 instances are all known, namely $LB = \text{the optimal solution for each instance, as the definition of } LB \text{ is shown in Section 2.2.}$

In addition, in most of the real-world problems, the optimal solutions involve some waste regions. Therefore, it is very interesting and useful to test FH on this type of instances (namely non-zero-waste instances) to verify the performance of different algorithms. In fact, the floating-point data sets with similarly dimensioned rectangles (data set CX: 50cx ~ 15000cx) to vastly differing dimensions (data set Path: Path1 ~ Path5t) in Valenzuela and Wang (2001) belong to the non-zero-waste instances because they are transformed by rounding near integer (Alvarez-Valdes et al., 2008) from float data to integer data. These instances vary from 25 to 5,000 rectangles; their optimal solutions are unknown. Large-scale non-zero-waste data set ZDF (zdfl $X \sim zdfl9$) is included, seven extra large-scale

---

**FastHeuristic()**

Sort all unplaced rectangles by non-increasing ordering of perimeter size and obtain ordering $X;

\[
\text{best}_h \leftarrow \text{HeuristicPlacing}(X); // \text{best}_h \text{ denotes the best height so far}
\]

for $i \leftarrow 1 \text{ to } n-1$

for $j \leftarrow i+1 \text{ to } n$

Swap the order of rectangle $i$ and $j$ in the current ordering $X$ and obtain the new ordering $X'$;

\[
\text{current}_h \leftarrow \text{HeuristicPlacing}(X'); //\text{current}_h \text{ denotes the height in the new orderings } X' \;
\]

if current$_h < \text{best}_h$

\[
\text{best}_h \leftarrow \text{current}_h;
\]

$X \leftarrow \text{current}_h$;

else do not swap, namely $X$ does not change;

return best$_h$

---

Fig. 5. A fast heuristic algorithm.

3. Computational results

In order to verify the performance of FH, we compare it with other latest published heuristic and meta-heuristic algorithms by testing a large amount of benchmark problem instances from the literature. GA + BLF and SA + BLF (Hopper & Turton, 2001), HR (Zhang et al., 2006) and Best-fit (BF) (Burke et al., 2004) are very good algorithms, but they have been beaten by the latest algorithms, such as BF + metaheuristics and HRP for problem type RF. These latest algorithms are selected to be compared with FH because many RF type instances have been tested by them.

The benchmark instances on the two-dimensional orthogonal strip packing problem include 21 problem instances (data set C: C11 ~ C73) ranging from 16 to 197 items in Hopper and Turton (2001) and 13 problem instances (data set N: N1 ~ N13) randomly generated by Burke et al. (2004), and especially a large instance involving 3152 items is given. The seven extra large-scale instances (data set CX: 50cx ~ 15000cx) proposed by Pinto and Oliveira (2005) are included. The problem scale of these instances varies from 50 to 15,000 rectangles. The optimal solution of the above 41 instances are all known, namely $LB = \text{the optimal solution for each instance, as the definition of } LB \text{ is shown in Section 2.2.}$

In addition, in most of the real-world problems, the optimal solutions involve some waste regions. Therefore, it is very interesting and useful to test FH on this type of instances (namely non-zero-waste instances) to verify the performance of different algorithms. In fact, the floating-point data sets with similarly dimensioned rectangles (data set CX: 50cx ~ 15000cx) to vastly differing dimensions (data set Path: Path1 ~ Path5t) in Valenzuela and Wang (2001) belong to the non-zero-waste instances because they are transformed by rounding near integer (Alvarez-Valdes et al., 2008) from float data to integer data. These instances vary from 25 to 5,000 rectangles; their optimal solutions are unknown. Large-scale non-zero-waste data set ZDF (zdfl $X \sim zdfl9$) is included, seven extra large-scale
non-zero-waste instances (zdf10 ~ zdf16) are generated by combining non-zero-waste instance gcut13 with zero-waste instances CX (see Appendix, especially, the scale of zdf16 is 75032 (n = 75032). The optimal solutions of the first five instances are known, the ones of the last 11 instances are unknown. (All data sets are available at http://59.77.16.8/Download.aspx#p4.)

FH coded in C++, was run on a 2 GHz Pentium 4 notebook with 2048 MB RAM. FH is allowed one run of 60 s. Since FH is a deterministic algorithm, it only runs once. BF + SA is the best algorithm among BF + metaheuristics including BF + TS, BF + SA and BF + GA, so we only select the best BF + SA for comparison. BF + SA was conducted on a 2 GHz Pentium 4 computer with 256 MB RAM. BF + SA is allowed one run of 60 s and the best solution (besth) is shown during 10 runs. HRP is designed to calculate the waste area of a packing. It is extended to solve the problem considered in this paper, so its original results slightly differ from the results obtained by this paper. HRP executable program obtained by Dr. Chen Duanbing can compute the height of each instances and runs faster. FH and HRP were run on the same computer and only run once. The solution (h) and the running time (time) are reported. The best solutions are bold-typed.

3.1. Computational results on problem type RF

3.1.1. Computational results on the data set C

For data set C, the computational results of BF + SA are directly taken from Burke Edmund et al. (2009). HRP and FH are run on the same machine and their computational results are reported in Table 1.

On this data set C, we observe that, BF + SA and HRP perform better than FH for small instances which are C13 and C33. However, FH outperforms BF + SA for large-scale instances. Both FH and HRP find the same solutions for large-scale instances (n > 49), but FH is faster than HRP.

3.1.2. Computational results on the data set N

The computational results of BF + SA for the data set N are also directly taken from Burke Edmund et al. (2009). The computational results of HRP and FH are reported in Table 2. On this data set N, as shown in Table 2, in nine instances marked by underline among the 13 instances under column “BF + SA”, FH obtains a smaller h than BF + SA. In four instances marked by italic among the 13 instances, FH is worse than BF + SA. In four instances marked by underline among the 13 instances under column “HRP”, FH obtains a smaller h than HRP. In two instances marked by italic among the 13 instances, FH is worse than HRP. For large-scale instances N12 and N13, FH can obtain better solutions in a short time than HRP. Moreover, FH finds the optimal solution of the largest scale instance N13.

3.1.3. Computational results on the extra large-scale data set CX

For the extra large-scale data set CX, Table 3 reports the computational results of HRP and FH, where “—” denotes FH cannot find a solution in 10000 s. FH can find the optimal solutions of most instances except 50cx and 100cx. Moreover, for instance 5000cx, the solution obtained by FH is better than that obtained by HRP in a very long time. So FH outperforms HRP for large-scale instances (n > 100).

3.1.4. Computational results on the non-zero-waste data set Nice and Path

The computational results on zero-waste instances show that FH outperforms BF + SA and HRP for large-scale instances. Now we test non-zero-waste data set Nice and Path and compare FH with BF + SA and HRP. The computational results are reported in Table 4, where “—” denotes the instance is not computed by BF + SA or the solution of the instance is not obtained by HRP within 10000 s.

From Table 4, it is observed that in nine instances marked by underline among the 12 instances (Nice1 ~ Nice6 and Path1 ~ Path6) under column “BF + SA”, FH finds a smaller h than

Table 1: Computational results on data set C.

<table>
<thead>
<tr>
<th>Instance</th>
<th>n</th>
<th>W</th>
<th>LB</th>
<th>BF + SA</th>
<th>HRP</th>
<th>FH</th>
</tr>
</thead>
<tbody>
<tr>
<td>C11</td>
<td>16</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>C12</td>
<td>17</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>0.19</td>
<td>0.01</td>
</tr>
<tr>
<td>C13</td>
<td>16</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>0.03</td>
<td>0.01</td>
</tr>
<tr>
<td>C21</td>
<td>25</td>
<td>40</td>
<td>15</td>
<td>16</td>
<td>0.27</td>
<td>0.02</td>
</tr>
<tr>
<td>C22</td>
<td>25</td>
<td>40</td>
<td>15</td>
<td>16</td>
<td>0.06</td>
<td>0.15</td>
</tr>
<tr>
<td>C23</td>
<td>25</td>
<td>40</td>
<td>15</td>
<td>16</td>
<td>0.05</td>
<td>0.15</td>
</tr>
<tr>
<td>C31</td>
<td>28</td>
<td>60</td>
<td>30</td>
<td>31</td>
<td>0.61</td>
<td>0.02</td>
</tr>
<tr>
<td>C32</td>
<td>29</td>
<td>60</td>
<td>30</td>
<td>31</td>
<td>0.61</td>
<td>0.02</td>
</tr>
<tr>
<td>C33</td>
<td>28</td>
<td>60</td>
<td>30</td>
<td>31</td>
<td>0.63</td>
<td>0.02</td>
</tr>
<tr>
<td>C41</td>
<td>49</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>0.23</td>
<td>0.11</td>
</tr>
<tr>
<td>C42</td>
<td>49</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>0.64</td>
<td>0.16</td>
</tr>
<tr>
<td>C43</td>
<td>49</td>
<td>60</td>
<td>60</td>
<td>60</td>
<td>1.61</td>
<td>0.06</td>
</tr>
<tr>
<td>C51</td>
<td>73</td>
<td>60</td>
<td>90</td>
<td>91</td>
<td>4.30</td>
<td>0.28</td>
</tr>
<tr>
<td>C52</td>
<td>73</td>
<td>60</td>
<td>90</td>
<td>91</td>
<td>1.34</td>
<td>0.03</td>
</tr>
<tr>
<td>C53</td>
<td>73</td>
<td>60</td>
<td>90</td>
<td>91</td>
<td>4.30</td>
<td>0.13</td>
</tr>
<tr>
<td>C61</td>
<td>97</td>
<td>80</td>
<td>120</td>
<td>122</td>
<td>9.84</td>
<td>1.08</td>
</tr>
<tr>
<td>C62</td>
<td>97</td>
<td>80</td>
<td>120</td>
<td>121</td>
<td>8.38</td>
<td>1.09</td>
</tr>
<tr>
<td>C63</td>
<td>97</td>
<td>80</td>
<td>120</td>
<td>121</td>
<td>9.94</td>
<td>1.07</td>
</tr>
<tr>
<td>C71</td>
<td>196</td>
<td>160</td>
<td>240</td>
<td>244</td>
<td>61.48</td>
<td>13.91</td>
</tr>
<tr>
<td>C72</td>
<td>197</td>
<td>160</td>
<td>240</td>
<td>244</td>
<td>58.91</td>
<td>11.64</td>
</tr>
<tr>
<td>C73</td>
<td>196</td>
<td>160</td>
<td>240</td>
<td>245</td>
<td>62.9</td>
<td>15</td>
</tr>
</tbody>
</table>

0 means the running time is less than 0.01 second.

Table 2: Computational results on data set N.

<table>
<thead>
<tr>
<th>Instance</th>
<th>n</th>
<th>W</th>
<th>LB</th>
<th>Besth</th>
<th>BF + SA</th>
<th>HRP</th>
<th>FH</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>10</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>0.03</td>
<td>0</td>
</tr>
<tr>
<td>N2</td>
<td>20</td>
<td>30</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>0.02</td>
<td>0</td>
</tr>
<tr>
<td>N3</td>
<td>30</td>
<td>30</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>0.02</td>
<td>0</td>
</tr>
<tr>
<td>N4</td>
<td>40</td>
<td>80</td>
<td>80</td>
<td>80</td>
<td>80</td>
<td>2.09</td>
<td>0.3</td>
</tr>
<tr>
<td>N5</td>
<td>50</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>4.08</td>
<td>0.08</td>
</tr>
<tr>
<td>N6</td>
<td>50</td>
<td>50</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>5.55</td>
<td>0.03</td>
</tr>
<tr>
<td>N7</td>
<td>70</td>
<td>80</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>9.0</td>
<td>0.13</td>
</tr>
<tr>
<td>N8</td>
<td>80</td>
<td>100</td>
<td>80</td>
<td>80</td>
<td>80</td>
<td>6.78</td>
<td>0.23</td>
</tr>
<tr>
<td>N9</td>
<td>100</td>
<td>50</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>23.27</td>
<td>0.14</td>
</tr>
<tr>
<td>N10</td>
<td>200</td>
<td>70</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>61.19</td>
<td>0.55</td>
</tr>
<tr>
<td>N11</td>
<td>300</td>
<td>70</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>67.69</td>
<td>1.48</td>
</tr>
<tr>
<td>N12</td>
<td>500</td>
<td>100</td>
<td>300</td>
<td>300</td>
<td>300</td>
<td>85.72</td>
<td>5.63</td>
</tr>
<tr>
<td>N13</td>
<td>3152</td>
<td>640</td>
<td>960</td>
<td>964</td>
<td>964</td>
<td>2743.48</td>
<td>1.91</td>
</tr>
</tbody>
</table>

Table 3: Computational results on the extra large-scale data set CX.

<table>
<thead>
<tr>
<th>Instance</th>
<th>n</th>
<th>W</th>
<th>LB</th>
<th>BF + SA</th>
<th>HRP</th>
<th>FH</th>
</tr>
</thead>
<tbody>
<tr>
<td>50cx</td>
<td>50</td>
<td>40</td>
<td>400</td>
<td>600</td>
<td>615</td>
<td>10.09</td>
</tr>
<tr>
<td>100cx</td>
<td>100</td>
<td>40</td>
<td>400</td>
<td>600</td>
<td>615</td>
<td>47.88</td>
</tr>
<tr>
<td>500cx</td>
<td>500</td>
<td>40</td>
<td>400</td>
<td>600</td>
<td>615</td>
<td>37.31</td>
</tr>
<tr>
<td>1000cx</td>
<td>1000</td>
<td>40</td>
<td>400</td>
<td>600</td>
<td>607</td>
<td>86.08</td>
</tr>
<tr>
<td>5000cx</td>
<td>5000</td>
<td>40</td>
<td>400</td>
<td>600</td>
<td>607</td>
<td>925.37</td>
</tr>
<tr>
<td>10000cx</td>
<td>10000</td>
<td>40</td>
<td>400</td>
<td>600</td>
<td>607</td>
<td>600</td>
</tr>
<tr>
<td>15000cx</td>
<td>15000</td>
<td>40</td>
<td>400</td>
<td>600</td>
<td>607</td>
<td>600</td>
</tr>
</tbody>
</table>

BF + SA. In three instances marked by italic among these 12 instances, FH is worse than BF + SA. In seven instances marked by underline among the same 12 instances (Nice1 ~ Nice6 and Path1 ~ Path6) under column “HRP”, FH finds a smaller h than HRP. In five instances marked by italic among these 12 instances, FH is worse than HRP.

For Nice1 ~ Nice5t and Path1t ~ Path5t, FH outperforms HRP for Nice1t, Nice2t, Nice5t and Path1t, Path2t and Path5t in addition, HRP cannot obtain the solutions of Nice5t and Path5t within 10000 s. Moreover, FH runs faster than HRP. By closer inspection, this kind of bad cases still occurs in problem instances involving instances of small scale. It has shown that FH outperforms BF + SA and HRP on large-scale non-zero-waste instances.

### 3.2. Computational results for problem type OF

The above computational results show that FH outperforms the current excellent algorithms BF + SA and HRP for problem type RF. In this section, it is of interest to validate whether FH is still efficient to solve the problem without rotation of items, namely, the items are placed by fixed direction (Problem type OF). To extend FH to solve OF, we modify the former algorithm by replacing “sort the rectangles in descending order of their perimeter” with “sort the rectangles in descending order of their height” in FastHeuristic() (see Fig. 5) and fix the direction of the rectangles. To verify the performance of FH, we compare FH with GRASP and SVC. To our knowledge, GRASP and SVC are the best algorithms for problem type OF so far. Especially, SVC performs better for large-scale instances (Belov et al., 2008). We test FH on these large-scale instances (n > 200) in data sets N, Nice, Path and the extra large-scale instances CX. Large-scale data sets Nice1t, Nice2t, Nice5t and Path1t, Path2t and Path5t which include 10 instances respectively are included. These 60 instances belong to non-zero-waste instance because of the transformation from float to integer data. In addition, some extra large-scale non-zero-waste instances are included. The computational results of GRASP for all the instances are obtained by the executable program of Prof. Ramon Alvarez-Valdes, and the computational results of SVC are obtained by the executable program of Dr. G. Belov. GRASP and SVC are only run once. The time limit of GRASP, SVC and FH is 60 s, and gap is \((100 \times (h - LB)/LB)\) for GRASP, SVC and FH.

Table 5 shows the computational results of GRASP, FH and SVC on large-scale instances \((n > 200)\) in the above data sets N, CX, Nice and Path. For large-scale data sets, it is observed from Table 5 that FH outperforms GRASP and SVC in eight instances marked by bold. However, GRASP and SVC cannot find any better solution than FH. So for large-scale instances, FH outperforms GRASP and SVC. Especially, FH finds the optimal solution of 1000cx.

Table 6 shows the computational results of GRASP, SVC and FH on large-scale instances \((n > 200)\) in the above data sets, and the computational results of FH are obtained by the executable program of Dr. G. Belov. GRASP and SVC are only run once. The time limit of GRASP, SVC and FH is 60 s, and gap is \((100 \times (h - LB)/LB)\) for GRASP, SVC and FH.

For problem type RF, it is observed from Table 7 that FH can find better solutions than HRP for zdf1 ~ zdf10, zdf8 and zdf9. However, FH cannot obtain the solution of zdf10 ~ zdf16 within 10000 s, while HRP is superior for zdf6 and zdf7.

For problem type OF, FH can outperform GRASP and SVC for zdf1 ~ zdf5 and zdf8 ~ zdf16, while GRASP and SVC are superior for zdf6 and zdf7. Especially, FH can find optimal solutions of zdf1 ~ zdf5, and zdf13, while SVC only find one optimal solution for zdf5, and GRASP fails to obtain any optimal solution. In addition, we can observe several interesting phenomena:

(1) For instances zdf6 and zdf7, FH fails to find better solution than HRP for problem type RF; the same result occurs in problem type OF. FH does not perform well.

(2) For instances zdf11, zdf12 and zdf13, FH for problem type OF actually finds better solution than FH for RF. The same case occurs for Path2t and Path5t.
For problem type RF, FH can find the optimal solutions of extra large-scale instances zdf14, zdf15 and zdf16. It is worth doing more test whether FH can find the optimal solutions for other extra large-scale non-zero-waste instances or not.

Table 8 reports the computational results of SC, SCR and our proposed algorithm for 70 Nice and Path instances. The three algorithms are layer-based heuristics. The results of SC(R) are taken from Ortmann et al. (2010), performing on a Windows XP PC with a 3.0 GHz Intel Core 2 Duo CPU and 4 GB RAM. Our algorithm is run on a Windows XP PC with a 1.73 GHz Intel(R) Pentium(R) M processor and 504 MB RAM. Although we do not test more instances because our aim is to compare our algorithm with these algorithms with good solution quality, from Table 8, we can observe that our algorithm is very efficient.

Fig. 6 gives four placements for N13 and zdf13 for problem type RF and OF respectively. We can observe that the placements for problem type OF are still much dense. For zdf13, it is very interesting that FH first stacks the rectangles (its width = 1).

In all, the experimental results are summarized as follows:

1. For problem type RF, FH does not perform well for very small instances; however, for the instances of scale greater than 200, FH outperforms the existing excellent algorithms BF + SA and HRP.
2. Although FH is not specially designed for problem type OF, it is still very efficient while solving them. FH outperforms the current excellent algorithms GRASP and SVC for the considered large-scale instances (n > 200). It would be interesting if GRASP and SVC also produce good solutions on these instances given that more computational time is allowed.
3. FH can efficiently solve zero-waste and non-zero-waste instances.
4. BF + SA, SVC and GRASP belong to randomized algorithm, their performance depends on the settings of the parameters. Especially, GRASP needs learning according to some instances, and then determines the parameter, so it is more complex. HRP has no parameter setting, but it needs much more time to obtain a desirable solution. However, FH is a deterministic algorithm without setting any parameters, and it runs very fast and can obtain a desirable solution for a broad range of large-scale benchmark problem instances in a very short time.
FH performs well because FH can select one appropriate rectangle for a given position and a good data structure is used to save the current available positions, which helps to determine the lowest position fast. However, for some especial instances, for example, the dimensions of each rectangle are strictly different and these rectangles are packed with fixed direction, FA may perform worse because it may not be able to take the advantage of fitness value strategy.

3.3. Effect of sorting

FH is a deterministic algorithm, but it involves a sorting step. In order to assess the effect of the sorting step, we consider two sorting steps for RF type problem: one sorting step by perimeter is to sort the rectangles in descending order of their perimeter, another step by area is to sort the rectangles in descending order of their area. The computational results are reported in Fig. 7(1–6), where the horizontal ordinate denotes the scale of problem (n), the vertical ordinate denotes the gap. In Fig. 7, it is observed that the sorting step by perimeter is better than the sorting step by area for data sets C and N (see Fig. 7(1) and (5), respectively). The sorting step by area is better than the sorting step by perimeter for data sets CX, Path and ZDF (see Fig. 7(2), (4) and (6), respectively). The total average gap of the sorting step by perimeter is 1.41, and the total average gap by area is 1.93. It seems that the first sorting step is better than the second one; however, the total average gap of the first sorting step is 1.40, and the total average gap of the second is 1.28 if the exterminate instance N7 is not included. The second sorting step performs better for more data sets. Therefore, the two sorting steps almost have no differences. This shows that FH can obtain good results whether the sorting step is by perimeter or area. The sorting by perimeter is selected in this paper because it is more stable and performs better on average.

4. Conclusions

A fast layer-based heuristic algorithm for the large-scale orthogonal strip packing problems is presented in this paper. This algorithm inspired by nature is very simple and intuitive, and can solve the orthogonal strip packing problem efficiently. The computational results show that FH can compete with some latest metaheuristics in terms of both solution quality and execution time. Especially, it performs better for large-scale test problems. In addition, FH does not involve the selection of parameters. However, meta-heuristics often involve many parameters on whose selection their performance significantly depends. So FH may be of great practical value to the rational layout of rectangular objects in the engineering fields, such as the wood-, glass- and paper industries, and the ship building industry, textile and leather industry.

To our knowledge, this is the first paper that presents a simple and deterministic algorithm and has attempted to test such a broad range of large-scale benchmark problem instances from the literature and obtains better solutions for large-scale instances in a very short time when comparing with the latest heuristics and metaheuristics. However, there is still much room for improvement and investigation for the proposed algorithm. First, we can extend it to solve three-dimensional rectangular packing problems. Second, in the cases where the 60 s limit was reached, it would be interesting to know the number of pairs explored for some big instances.

Acknowledgments

The authors thank the anonymous referees for their invaluable comments and thank to Dr. Glenn Whitwell, Prof. Manuel Iori, Dr. C.L. Mumford and Dr. Graham Kendall for their help to provide test data sets for this paper. Special thanks to Prof. Ramon Alvarez-Valdes, Dr. G. Belov and Dr. Duanbing Chen for their executable programs of GRASP, SVC and HRP respectively. Special thanks also to Dr. Andreas Bortfeldt for his valuable help and data sets. This work has been supported by the National Nature Science Foundation of China (Grant No. 60773126) and the Province Nature Science Foundation of Fujian (Grant Nos. 2007J0037 and 2010J01362).
Appendix A

References


