World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:3, No:5, 2009

A New Heuristic Approach for the Stock-Cutting Problems

Stephen C. H. Leung, and Defu Zhang

***Abstract*—**This paper addresses a stock-cutting problem with rotation of items and without the guillotine cutting constraint. In order to solve the large-scale problem effectively and efficiently, we propose a simple but fast heuristic algorithm. It is shown that this heuristic outperforms the latest published algorithms for large-scale problem instances.

***Keywords*—**Combinatorial optimization, heuristic, large-scale, stock-cutting.

I. Introduction

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. 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 necessary to consider how to place goods on shelves. In the newspapers typesetting field, it is necessary to consider how to arrange articles and advertisements in pages. In the shipping industry, it is necessary to consider 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 necessary to consider how to accommodate a bunch of optical fibers in a pipe as small as possible. In VLSI floor planning industry, it is necessary to consider how to lay VLSI. These applications can be formalized as a cutting and packing problem with different constraints and objectives [1]. 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. On the other hand, there is a goal which is to produce a solution within a very short time. The later goal is especially important in the logistic fields because any delay will lead to a loss of customers. Therefore, it is usually very important to produce better-quality solutions in less time to meet the needs of industrial applications. In some industrial fields, the cutting or packing task is always done by skilled workers. However, due to a lack of material and the need of industrial applications, automated-packing algorithms have become more widely used in recent years [2]. For more algorithms or reviews on cutting and packing problems, the interested reader is referred to the literature [1, 3, 4].

In this paper, the two-dimensional orthogonal stock-cutting problem is considered without any guillotine constraint. This problem can be stated as follows: Given a rectangular sheet of a given width and an infinite height and a set of rectangles with arbitrary sizes, the orthogonal stock-cutting 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 ($1 \le i \le n$) respectively. A more formal statement for this problem is given in [5]. In this paper, we assume that the edges of each rectangle are parallel to the edges of the rectangular sheet, namely, orthogonal cuts. In addition, all rectangles must be placed into the rectangular sheet and are allowed to rotate 90 degree, namely, RF subtype according to [6]. This problem can be classified as a two-dimensional single large object placement problem according to [7].

The rest of this paper is organized as follows. In section II, inspired by the wall-building rule of bricklayers in daily life, we present a fast heuristic algorithm. Computational results are described in section III. Conclusions are summarized in section IV.

II. A Heuristic for Large-Scale Stock-Cutting Problem

In this section, a bricklaying heuristic strategy to explain the work idea of wall building is first introduced [8]. Then it is extended to design fast heuristics for the considered cutting problem.

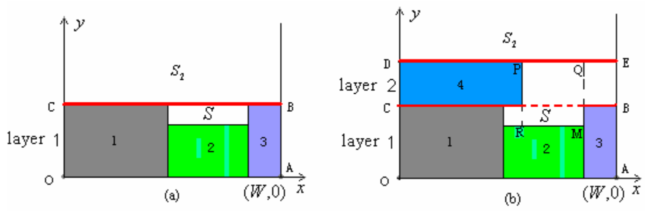

Fig. 1 The process of placing rectangles by bricklaying heuristic algorithm

The idea of bricklaying heuristic algorithm is to place rectangles by layer [8]. A new layer determined by a reference rectangle starts when the current layer cannot place more

Stephen C. H. Leung is with the Department of Management Sciences, City University of Hong Kong, Hong Kong (phone: 852-2788-8650; fax: 852-2788-8560; e-mail: mssleung@cityu.edu.hk).
Defu Zhang is with the Department of Computer Science, Xiamen University, Xiamen 361005, China (e-mail: dfzhang@xmu.edu.cn).

rectangles. The layers are divided by the reference line. For example, in Fig. 1(a), the reference rectangle 1 is placed into the position O and forms the layer 1. 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 make full use of 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 cutting 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.

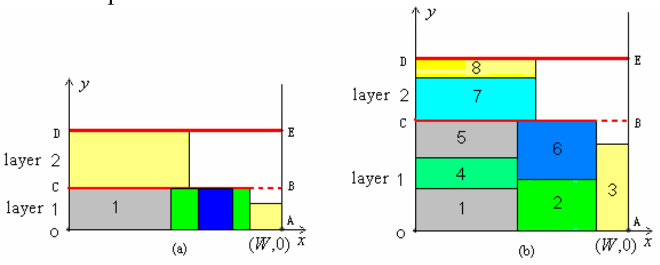

Fig. 2 The process of placing rectangles by proposed heuristic algorithm

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 edge. For example, some small rectangles have to be placed under the reference line BC defined by the rectangle 1 in Fig. 2(a). From Fig. 2(b), the height of the layer will be higher if the rectangles 1, 4 and 5 are stacked together; this will allow the large rectangles to be placed first. Indeed, the rectangles 2 and 6 cannot be placed under the reference line BC in Fig. 2(a), but they can be placed under the reference line BC in Fig. 2(b). Therefore, in step 1, we should stack the rectangle of the same edge until this layer's height exceeds the lower bound of solution *LB*, where $LB=\left\lceil \sum_{i=1}^{n} h_i w_i \Big/ W \right\rceil$. In step 2, it is very important to select one rectangle to be placed in the current available position. For an available position *p*, in the rectangle space determined by *p* there are two kinds of possible rectangular space *S* determined by position *p*. *S* has four corner positions which touch the placed rectangles, the corner positions marked by circle in Figs. 3(1) and 3(2). Let h_1 denote the height of the wall adjacent to the left of *S*, h_2 denote the height of the wall adjacent to the right of *S* (see Figs. 3(1) and 3(2)). The first case is $h_1 \geq h_2$. For this case, the unplaced rectangles should close h_1 to place. 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 one 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 one rectangle *R* marked by blue is placed into the position *p* and obtain 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 one rectangle *R* marked by blue is placed into the position *p* and obtain 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 one rectangle *R* marked by blue is placed into the position *p* and obtain 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 one rectangle *R* marked by blue is placed into the position *p* and obtain 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 close h_2 to place. Similarly, there are four kinds of possible available placements. For each placement in Figs. 3(2.1), (2.2), (2.3) and (2.4), the fitness value of the corresponding rectangle R is 3, 2, 1 and 0 respectively. The fitness value of the rectangle is $-\infty$ if that rectangle cannot be placed into the rectangle space S.

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. 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.

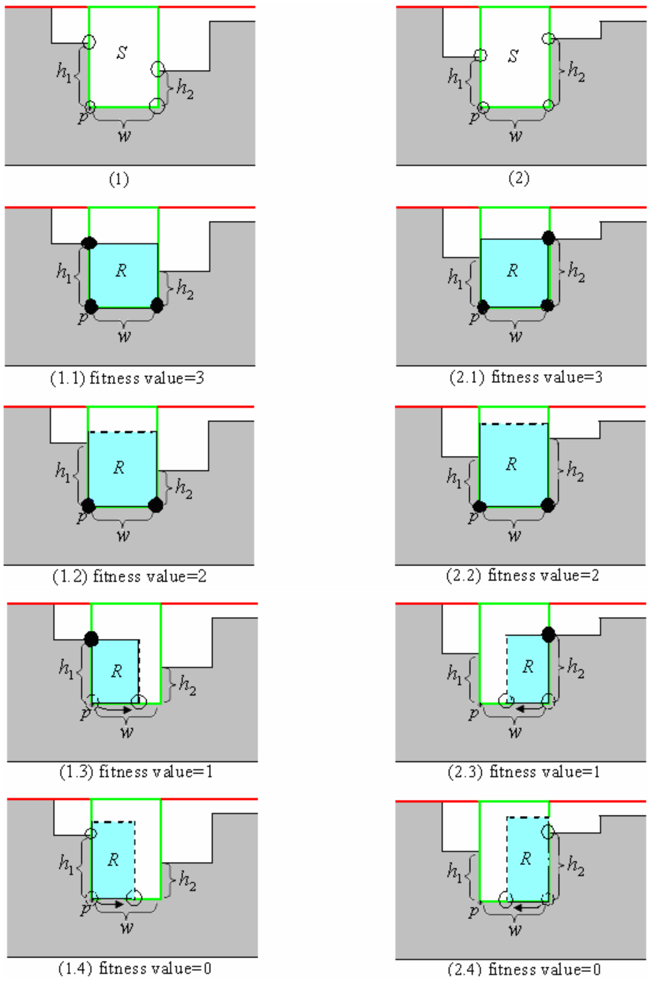

Fig. 3 Fitness value

```
HeuristicPlacing($X$)
  repeat
     select the first unplaced rectangle $r$ in the current rectangles sequence;
     if the long edge of the rectangle $r$ is not greater than $W$ then
        place its long edge of the rectangle $r$ along the current reference line;
        let the length of $r$'s long edge be $l$
     else
        place its short edge of the rectangle $r$ along the current reference line;
        let the length of $r$'s short edge be $l$
     the current reference rectangle is $r$
     the current reference line determined by $r$ is $L$, its height is $h$
     repeat
        select one unplaced rectangle $r_1$ whose one edge length equals to $l$
        stack $r_1$ above $r$ and let $r = r_1$;
        compute the height $h$ of the current reference line $L$
    until the layer height $h$ exceeds $LB$;
    repeat
        determine the available lowest position $p$ under $L$ and compute $h_1$, $h_2$;
        if $h_1 \geq h_2$ then
           compute the fitness value of each unplaced rectangles by the first case;
           select the rectangle $R$ with the maximum fitness value;
           place the rectangle $R$ by the first case;
        else
           compute the fitness value of each unplaced rectangles by the second case;
           select the rectangle $R$ with the maximum fitness value;
           place the rectangle $R$ by the second case;
    until no rectangles can be placed under the reference line $L$;
  until all rectangles are placed;
return $h$;
```

Fig. 4 A fast heuristic algorithm

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.

Since the performance of heuristic algorithm significantly depends on the placing ordering X of the rectangles, some research results have shown that the placing ordering of the rectangles affects the performance of the presented algorithm [5, 6, 9]. In this paper, we make use of a heuristic strategy for selecting an initial placing ordering, namely unplaced rectangles should be sorted by a non-increasing ordering of perimeter size before placing. According to this placing ordering, a rectangle with the maximum perimeter is given a higher priority to be placed so that the largest rectangle can be placed first. In addition, the way of strictly placing rectangles by their perimeter ordering does not correspond to the way of practical placing in the industry fields. Therefore, we can try different orderings to find a better solution, and then develop a fast heuristic (FH) algorithm (see Fig. 5) as follows.

```
FastHeuristic()
  Sort all unplaced rectangles by non-increasing ordering of
  perimeter size and obtain ordering $X$;
  besth ← HeuristicPlacing($X$); // besth denotes the best
  height so far
  for $i$ ← 1 to $n$–1 do
      for $j$ ← $i$+1 to $n$ do
          Swap the order of rectangle $i$ and $j$ in the current
          ordering $X$ and obtain the new ordering $X'$;
          currenth ← HeuristicPlacing($X'$); //currenth denotes
          the height in the new orderings $X'$ ;
          if currenth < besth then
              besth ← currenth;
              $X$ ← $X'$;
          else do not swap, namely $X$ does not change;
 return besth
```

Fig. 5 A fast heuristic algorithm

From Fig. 5, FH first sorts the unplaced rectangles by a non-increasing ordering of perimeter size, then HeuristicPlacing(X) is called once according to this perimeter ordering X and obtains an initial best*h*. Then, FH executes the two for loops, for given i and j, the algorithm first swaps the order of rectangle i and j in the current orderings X and obtain a new orderings X', then computes current*h* in the new orderings X'. If current*h* is less than best*h*, then update best*h* and X. Otherwise, do not swap, namely keep X unchanged. Repeat this process until the two for loops are finished. For the given i and j, if swapping them can improve the height, then executes this swap, so FH makes use of the idea of greedy search. However, it does not use real greedy search because FH will exterminate after the two for loops are finished.

III. 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 [9], HR [5] and Best fit (BF) [10] 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 stock-cutting problem include 21 problem instances (data set C: C11~C73) ranging from 16 to 197 items in [9] and 13 problem instances (data set N: N1~N13) randomly generated by [10], and especially a large-scale instance involving 3152 items is given. The 7 extra large-scale instances (data set CX: 50cx~15000cx) proposed by [11] 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*=the optimal solution for each instance.

FH coded in C++, was run on a 2GHz Pentium 4 notebook with 2048MB RAM. FH is allowed one run of 60 seconds. Since FH is a deterministic algorithm, it only runs once. BF+SA is the best algorithm among BF+TS, BF+SA and BF+GA, so we only select the best BF+SA for comparison. BF+SA was conducted on a 2GHz Pentium 4 computer with 256MB RAM. BF+SA is allowed one run of 60 seconds and the best solution (best*h*) is shown during 10 runs. HRP is designed to calculate the waste area of a packing; it is extended to solve the problem considered by this paper, so its original results slightly differ from the results obtained by this paper. HRP executable program from the authors in [12] 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.

A. Computational Results

1. Computational results on the data set C

For data set C, the computational results of BF+SA are directly taken from [13]. HRP and FH are run on the same machine and their computational results are reported in Table I.

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. FH and HRP find the same solutions for large-scale instances (n > 49), but FH is faster than HRP.

TABLE I
COMPUTATIONAL RESULTS ON DATA SET C

Instance				BF+SA	HRP		FH	
	n	*W*	*LB*	best*h*	*h*	*time*	*h*	*time*
C11	16	20	20	20	20	0.03	20	0[*]
C12	17	20	20	20	20	0.19	20	0.01
C13	16	20	20	***20***	***20***	0.03	21	0
C21	25	40	15	16	***15***	0.27	16	0.02
C22	25	40	15	16	**15**	0.06	**15**	0
C23	25	40	15	16	**15**	0.05	**15**	0
C31	28	60	30	31	31	0.61	31	0.02
C32	29	60	30	31	31	0.61	31	0.02
C33	28	60	30	***31***	***31***	0.63	32	0.02
C41	49	60	60	61	61	1.84	61	0.16
C42	49	60	60	61	***60***	0.23	61	0.11
C43	49	60	60	61	61	2.02	61	0.08
C51	73	60	90	91	91	4.30	91	0.28
C52	73	60	90	91	**90**	1.34	**90**	0
C53	73	60	90	92	**91**	4.3	**91**	0.3
C61	97	80	120	122	**121**	9.84	**121**	0.83
C62	97	80	120	121	**121**	8.38	**121**	0.89
C63	97	80	120	122	**121**	9.94	**121**	0.76
C71	196	160	240	244	**241**	61.48	**241**	13.91
C72	197	160	240	244	**241**	58.91	**241**	11.64
C73	196	160	240	245	**241**	62.9	**241**	15

[*]: 0 means the running time is less than 0.01 second

2. Computational results on the data set N

The computational results of BF+SA for the data set N are directly taken from [15]. The computational results of HRP and FH are reported in Table II. On this data set N, as shown in Table II, in 9 instances marked by underline among the 13 instances under column "BF+SA", FH obtains a smaller *h* than BF+SA. In 2 instances marked by italic among the 13 instances, FH is worse than BF+SA. In 4 instances marked by underline among the 13 instances under column "HRP", FH obtains a smaller *h* than HRP. In 2 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.

TABLE II
COMPUTATIONAL RESULTS OF THE DATA SET N

instance				BF+SA	HRP		FH	
	n	*W*	*LB*	best*h*	*h*	*time*	*h*	*time*
N1	10	40	40	40	40	0.03	40	0
N2	20	30	50	***50***	*51*	0.023	52	0
N3	30	30	50	51	51	0.61	51	0.02
N4	40	80	80	*82*	***81***	2.09	83	0.03
N5	50	100	100	103	**102**	4.08	**102**	0.08
N6	60	50	100	102	102	5.55	**101**	0.03
N7	70	80	100	104	**102**	9.0	**102**	0.13
N8	80	100	80	82	**81**	6.78	**81**	0.23
N9	100	50	150	152	152	23.27	**151**	0.14
N10	200	70	150	152	**151**	61.19	**151**	0.55
N11	300	70	150	153	**151**	67.69	**151**	1.48
N12	500	100	300	306	305	85.72	**301**	5.63
N13	3152	640	960	964	972	2743.48	**960**	1.91

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

For the extra large-scale data set CX, Table III reports the computational results of HRP and FH, where "—" denotes HRP cannot find solution in 10000 seconds. 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 ($n > 100$).

TABLE III
COMPUTATIONAL RESULTS ON THE EXTRA LARGE-SCALE DATA SET CX

Instance				HRP		FH	
	n	*W*	*LB*	*h*	*time*	*h*	*time*
50cx	50	400	600	***615***	10.09	624	0.33
100cx	100	400	600	***615***	47.88	619	3.56
500cx	500	400	600	611	87.31	**600**	2.00
1000cx	1000	400	600	607	86.08	**600**	0.02
5000cx	5000	400	600	607	9256.37	**600**	0.05
10000cx	10000	400	600	—	—	**600**	0.05
15000cx	15000	400	600	—	—	**600**	0.06

IV. CONCLUSION

A fast heuristic algorithm for the large-scale orthogonal stock-cutting problem is presented in this paper. This algorithm inspired by nature is very simple and intuitive, and can solve the orthogonal stock-cutting problem efficiently. The computational results have shown that FH can compete with some latest meta-heuristics 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 so desirable solutions for large-scale instances in a very short time when comparing with the latest heuristics and metaheuristics. Future work is to further improve the performance of this algorithm and extend it to solve three-dimensional rectangular packing problems.

ACKNOWLEDGMENT

The authors would like to thank 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. 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 no. 2007J0037) and academician start-up fund (Grant No. X01109) and 985 information technology fund (Grant No. 0000-X07204) in Xiamen University.

REFERENCES

[1] A. Lodi, S. Martello, and D. Vigo, “Recent advances on two dimensional bin packing problems,” Discrete Applied Mathematics, vol. 123, pp. 379–396, 2002.

[2] Z. Li, and V. Milenkovic, “Compaction and separation algorithms for non-convex polygons and their applications,” European Journal of Operational Research, vol. 84, pp. 539–561, 1995.

[3] K. A. Dowsland, and W. B. Dowsland, “Packing problems,” European Journal of Operational Research, vol. 56, pp. 2–14, 1992.

[4] José Fernando Oliveira, and Gerhard Wäscher, “Cutting and Packing,” European Journal of Operational Research, vol. 183, pp. 1106–1108, 2007.

[5] Defu Zhang, Yan Kang, and Ansheng Deng, “A new heuristic recursive algorithm for the strip rectangular packing problem,” Computers and Operations Research, vol. 33, pp. 2209–2217, 2006.

[6] A. Bortfeldt, “A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces,” European Journal of Operational Research, vol. 172, pp. 814–837, 2006.

[7] Gerhard Wäscher, Heike Haußner, and Holger Schumann, “An improved typology of cutting and packing problems,” European Journal of Operational Research, vol. 183, pp. 1109–1130, 2007.

[8] Defu Zhang, Shui-hua Han, and Wei-guo Ye, “A bricklaying heuristic algorithm for the orthogonal rectangular packing problem,” Chinese Journal of Computers, vol. 23, pp. 509–515, 2008.

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

[10] E. K. Burke, G. Kendall, and G. Whitwell, “A new placement heuristic for the orthogonal stock-cutting problem,” Operations Research, vol. 52, pp. 655–671, 2004.

[11] E. Pinto, and J. F. Oliveira, “Algorithm based on graphs for the non-guillotinable two-dimensional packing problem,” Second ESICUP Meeting, Southampton, 2005.

[12] Wenqi Huang, Duanbing Chen, and Ruchu Xu, “A new heuristic algorithm for rectangle packing,” Computers and Operations Research, vol. 34, pp. 3270–3280, 2007.

[13] E. K. Burke, G. Kendall, and G. Whitwell, “Metaheuristic enhancements of the best-fit heuristic for the orthogonal stock cutting problem,” Computer science technical report no. NOTTCS-TR-SUB-0605091028-4370, University of Nottingham, 2006.