Available online at www.sciencedirect.com
ScienceDirect
Computers & Operations Research 34 (2007) 3270–3280
computers & operations research
www.elsevier.com/locate/cor

A new heuristic algorithm for rectangle packing

Wenqi Huang, Duanbing Chen*, Ruchu Xu

College of Computer Science, Huazhong University of Science and Technology, Wuhan 430074, China

Available online 3 February 2006

Abstract

The rectangle packing problem often appears in encasement and cutting as well as very large-scale integration design. To solve this problem, many algorithms such as genetic algorithm, simulated annealing and other heuristic algorithms have been proposed. In this paper, a new heuristic algorithm is recommended based on two important concepts, namely, the corner-occupying action and caving degree. Twenty-one rectangle-packing instances are tested by the algorithm developed, 16 of which having achieved optimum solutions within reasonable runtime. Experimental results demonstrate that the algorithm developed is fairly efficient for solving the rectangle packing problem.

Keywords: Rectangle packing; Heuristic algorithm; Caving degree; Corner-occupying action

1. Introduction

The rectangle packing problem often appears in encasement and cutting as well as VLSI design. To solve this problem, various algorithms based on different strategies have been suggested. In general, these algorithms can be classified into two major categories: non-deterministic algorithms and deterministic algorithms. The crux of non-deterministic algorithms, such as simulated annealing and the genetic algorithm [1,2], is to design data structures that can represent the geometric relationships among the rectangles, e.g., the O-tree [3,4], B*-tree [5], SP (sequence pair) [6–8], BSG (bounded sliceline grid) [9], CBL (corner block list) [10,11], TCG (transitive closure graph) [12,13], CS (corner sequence) [14], etc. The key aspect of deterministic algorithms is to determine the packing rules, such as the *less flexibility first principle* [15]. In addition, a number of effective quasi-human heuristic algorithms is proposed [16,17].

Inspired by the above approaches, a new heuristic rectangle packing algorithm (HRP) is proposed in this paper. The objective is to maximize the area usage of the box. The key issue of this algorithm is that the rectangle packed into the box always occupies a corner, even a cave, where possible. In this way, the rectangles will be close to each other wisely, and the spare space is decreased. Compared with those in literature, the results from HRP are greatly improved. Of the 21 rectangle-packing instances provided by Hopper and Turton [2], 16 achieved optimum solutions by HRP and 2 achieved optimum solutions by algorithm in [15]. Experimental results show that the HRP is rather efficient in solving the rectangle packing problem.

* Corresponding author. Tel.: +86 27 87552247; fax: +86 838 5152127.
E-mail address: hustcdb@yahoo.com.cn (D. Chen).

0305-0548/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2005.12.005

2. Problem description

An empty box B_0 of width w_0 and height h_0 is given. And there is a series of rectangles R_i $(i = 1, 2, \ldots, n)$ of width w_i and height h_i. In the plane rectangular coordinates, the bottom left of the box is placed at (0, 0) with its four sides parallel to X- and Y-axis, respectively. The objective is to maximize the area usage of the box, that is, to maximize the total area of the rectangles which have already been packed into the box. The constraints for packing rectangles are as follows:

1. Each edge of a rectangle should be parallel to an edge of the box.
2. There should be no overlapping for any two rectangles, i.e., the overlapping area is zero.

This version of packing is also known as "knapsack packing" since a subset of the available rectangles is so packed as to maximize the area used.

To describe the mathematical formulation, we give several notes about the problem.

1. We use z_i to denote whether the rectangle R_i is packed in or not. $z_i = 1$ if the rectangle R_i has been packed into the box, whereas $z_i = 0$ if it remains outside the box.
2. The bottom left and top right coordinates of rectangle R_i are (x_{li}, y_{li}) and (x_{ri}, y_{ri}), respectively.

The mathematical formulation can be stated as follows:

$$\text{maximize} \quad \sum_{i=1}^{n} w_i h_i z_i \tag{1}$$

subject to

$$\max[x_{li} - x_{rj}, x_{l_j} - x_{ri}, y_{li} - y_{rj}, y_{lj} - y_{ri}] z_i z_j \geqslant 0, \quad i, j = 1, 2, \ldots, n, \ i \neq j, \tag{2}$$

$$(x_{ri} - x_{li} = w_i) \wedge (y_{ri} - y_{li} = h_i) \quad \text{or} \quad (x_{ri} - x_{li} = h_i) \wedge (y_{ri} - y_{li} = w_i), \quad i = 1, 2, \ldots, n, \tag{3}$$

$$0 \leqslant x_{li} \leqslant w_0, 0 \leqslant x_{ri} \leqslant w_0 \quad \text{and} \quad 0 \leqslant y_{li} \leqslant h_0, \ 0 \leqslant y_{ri} \leqslant h_0, \quad i = 1, 2, \ldots, n, \tag{4}$$

$$z_i \in \{0, 1\}, \quad i = 1, 2, \ldots, n. \tag{5}$$

Eq. (2) implies there is no overlapping between two rectangles R_i and R_j; (3) implies R_i is located horizontally or vertically; (4) implies R_i is completely in the box B_0.

Note that, if rectangle R_i is not packed in $(z_i = 0)$, constraint (2) is automatically satisfied. For more details of mathematical formulation, the reader is referred to [18,19].

3. Description of the algorithm

3.1. Main idea

If some rectangles have been packed into the box without overlapping, the question is which one is the best candidate of the remainder, and which position is the best one to be filled? In this paper, we pack the rectangle according to the following packing principle: the rectangle to be packed into the box always occupies a corner, and the caving degree of the packing action should be as large as possible. So, the rectangles are close to each other wisely, and the area usage of the box improves consequently.

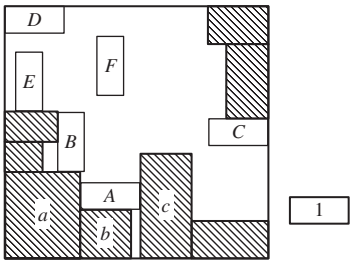

Fig. 1. Corner-occupying action.

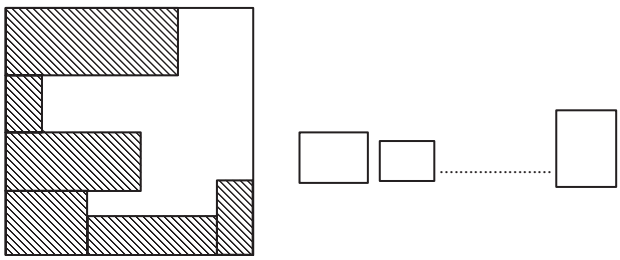

Fig. 2. Configuration.

3.2. Fundamental conceptions and strategies

(1) *Corner-occupying action* (COA): A packing action is called a *corner-occupying action* (COA),[1] if the rectangle to be packed touches two previously packed rectangles including the box,[2] (we can regard the 4 sides of the box as 4 rectangles of very small height which have already been put into the box at the original time) and the touching lengths are longer than zero. The rectangle to be packed occupies a corner formed by those two previously packed rectangles. A COA is called a feasible one, if the rectangle to be packed does not overlap any previously packed rectangle nor exceeds the box boundary. So, if a rectangle is packed into the box according to a feasible COA, all the constraints on packing are satisfied. We use tuple (x, y, Ori, R) to represent a COA, where R is the rectangle related to the corresponding COA, x and y are the coordinates of the bottom left of the rectangle R, Ori is the packing orientation ($Ori = 0$ if packed horizontally, while $Ori = 1$ if packed vertically). For instance, in Fig. 1, the shadowy rectangles have been packed, and the rectangle marked "1" is outside the box. The packing action is a feasible COA, if rectangle "1" is situated at place A, B, C or D; it is not a COA if situated at place E or F.

In particular, a COA is called a cave-occupying action, if the rectangle related to this COA not only occupies a corner, but also touches some other previously packed rectangles including the box. For instance, in Fig. 1, if rectangle "1" is situated at place A, it occupies the corner formed by rectangles a and b. Furthermore, it touches rectangle c. Thus, rectangle "1" occupies a cave formed by rectangles a, b and c, and this COA for packing rectangle "1" is a cave-occupying action.

(2) *Configuration*: Fig. 2 shows a configuration. Some rectangles have been packed into the box without overlapping and some remain outside the box. A configuration is called an initial one if there is no rectangle in the box. A configuration is called an end one if all n rectangles have already been packed into the box without overlapping or, none of the remaining rectangles can be packed in.

---

[1] COA is a developed form of corner occupying packing move (COPM) [15]. The scope of COA is wider than that of COPM, that is, a COPM must be a COA, but a COA may not be a COPM, e.g., in Fig. 1, if rectangle "1" is situated at place B, the packing action is a COA, but not a COPM.

[2] The principle of packing a rectangle to touch two previously packed rectangles is called *gapless packing* or *normalized packing* [20].

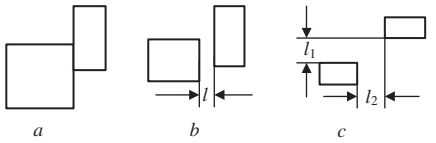

Fig. 3. Distance between two rectangles.

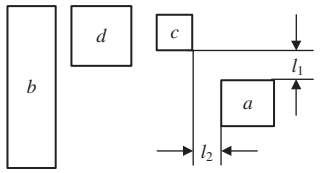

Fig. 4. Distance between one rectangle and several rectangles.

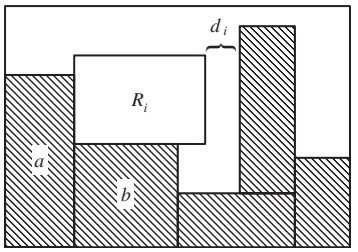

Fig. 5. Caving degree of COA.

(3) *Distance between two rectangles*: For two given rectangles R_i and R_j, the distance d_{ij} between them can be defined as

$$d_{ij} = \max\left(|x_{ci} - x_{cj}| - \frac{W_i + W_j}{2}, 0\right) + \max\left(|y_{ci} - y_{cj}| - \frac{H_i + H_j}{2}, 0\right), \tag{6}$$

where (x_{ci}, y_{ci}) and (x_{cj}, y_{cj}) are the central coordinates of the rectangles R_i and R_j, respectively.

$$W_i = x_{ri} - x_{li}, \quad H_i = y_{ri} - y_{li}, \quad W_j = x_{rj} - x_{lj}, \quad H_j = y_{rj} - y_{lj},$$

where (x_{li}, y_{li}) and (x_{ri}, y_{ri}) are the coordinates of the bottom left and top right of the rectangle R_i, respectively.

(x_{lj}, y_{lj}) and (x_{rj}, y_{rj}) are the coordinates of the bottom left and top right of the rectangle R_j, respectively.

This distance is an extension of Manhattan distance between two points.

For instance, as shown in Fig. 3a–c, the distance between two rectangles is zero, l and $l_1 + l_2$, respectively.

(4) *Distance between one rectangle and several rectangles*: For a given rectangle R and a set of rectangles $\{R_i | \ i = 1, 2, \ldots, m\}$, let the distance between R and R_i be d_i $(i = 1, 2, \ldots, m)$. The distance between rectangle R and m rectangles $R_1, R_2, \ldots, R_m$ is defined as $\min_{i\in\{1,2,3,\ldots,m\}}(d_i)$. For instance, as shown in Fig. 4, the distance between rectangle a and rectangles b, c, d is $l_1 + l_2$.

(5) *Caving degree of COA and maximum caving degree first strategy*: As shown in Fig. 5, rectangle R_i is packed into the box according to a feasible COA, and the distance between rectangle R_i and all the previously packed rectangles (including the box but excluding the rectangles a and b that form this corner) is d_i, the caving degree C_i of this COA can be defined as

$$C_i = 1 - \frac{d_i}{\sqrt{w_i h_i}}, \tag{7}$$

where w_i and h_i are the width and height of R_i, respectively.

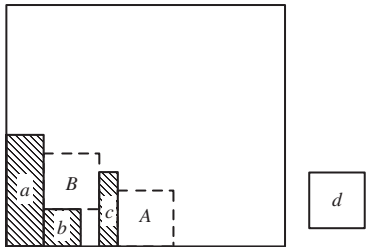

Fig. 6. An example for caving degree.

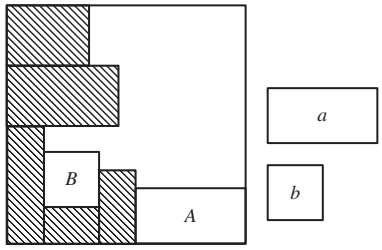

Fig. 7. Aspect ratio.

Actually, the caving degree reflects the nearness between the rectangle to be packed and its nearest rectangle (excluding the rectangles that form the corner). According to the definition, the caving degree of a COA is no greater than 1. In particular, it equals 1 when the corresponding rectangle occupies a cave formed by three previously packed rectangles; it may also be smaller than zero when $d_i > \sqrt{w_i h_i}$. In the packing process, we always select the COA with the largest caving degree, and pack the corresponding rectangle into the box in the corresponding position and orientation.

For instance, in Fig. 6, the dimensions of rectangles a, b, c, d and the box are (2, 6), (2, 2), (1, 4), (3, 3) and (15, 13), respectively. The rectangles a, b, c have been packed in, and the rectangle d is outside the box. If rectangle d is packed at the corner formed by rectangle c and one side of the box, the distance between rectangle d and its nearest rectangle b is 2, then the caving degree of this COA is equal to $1 - \frac{2}{\sqrt{3\times 3}} = \frac{1}{3}$. If rectangle d is packed at the corner formed by rectangles a and b, it not only occupies the corner formed by a and b, but also touches rectangle c. So, the nearest rectangle to d (excluding a and b) is c, then the caving degree of the corner-occupying action (to occupy the corner formed by a and b) is equal to $1 - \frac{0}{\sqrt{3\times 3}} = 1$.

Similarly, the caving degree for the COA to occupy the corner formed by b and c is also equal to 1.

(6) *Aspect ratio of a rectangle*: For a given rectangle R_i of width w_i and height h_i, the aspect ratio r_i of R_i can be defined as

$$r_i = \max(w_i, h_i)/\min(w_i, h_i). \tag{8}$$

In the packing process, if there are multiple COAs with equal and largest caving degree, we select such a COA whose corresponding rectangle has the largest aspect ratio. For example, in Fig. 7, the aspect ratios of rectangles a and b are 2 and 1, respectively. The caving degree is equal to 1 if rectangle a is packed at place A, and it is also equal to 1 if rectangle b is packed at place B. We pack rectangle a at place A since the aspect ratio of rectangle a is larger.

(7) *Precedence of point*: Let P_1 (x_1, y_1) and $P_2(x_2, y_2)$ be two points in the plane rectangular coordinates $o - xy$, P_1 has precedence over P_2 if $x_1 < x_2$, or if $x_1 = x_2$ and $y_1 < y_2$.

(8) Area usage of the box

$$\text{area usage} = \frac{\text{the total area of the rectangles packed in the box}}{\text{area of the box}} \tag{9}$$

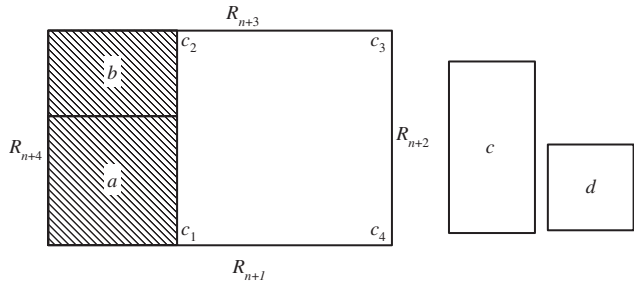

Fig. 8.

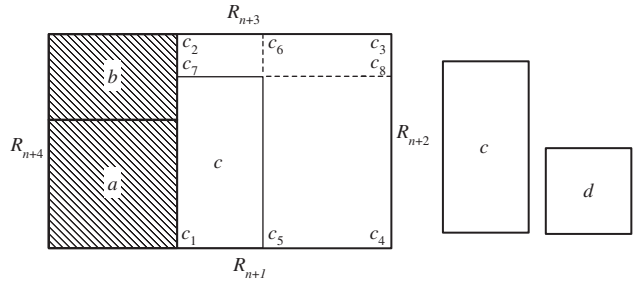

Fig. 9.

3.3. Sketch of the algorithm

(1) *Find possible corners and delete unavailable corners*: We regard the four sides of the box as four rectangles of very small heights R_{n+1}, R_{n+2}, R_{n+3}, R_{n+4}, and use tuple $(x, y, shape, R_{c1}, R_{c2})$ to represent a corner formed by rectangles R_{c1} and R_{c2} (Note that the rectangles R_{c1} and R_{c2} are not necessarily touching each other), Where x and y are x- and y-coordinate of the corner, respectively, and *shape* is the shape of the corner (there are four possible shapes: ⌊, ⌋, ⌉, ⌈).

When a rectangle R is packed into the box, the corners that are no longer available will be deleted and the new available corners formed by rectangle R and previously packed rectangles will be added. All corners formed by two packed rectangles can be found by using the relationship between two rectangles. The details for finding these corners are omitted here for space reasons. Readers are referred to [15].

For example, Fig. 8 illustrates an instance of four rectangles, rectangles a and b being packed. Before rectangles c and d are packed, there are four corners, denoted by c_1, c_2, c_3 and c_4. After rectangle c is packed at the corner c_1, as shown in Fig. 9 , corners c_1 and c_2 are no longer available because none of the rectangles outside the box can be packed at corner c_1 or c_2 without overlapping the previously packed rectangles. On the other hand, two new available corners c_5 and c_6 will be added. Corners c_7 and c_8 are unavailable.

(2) *Find all feasible positions for each unpacked rectangle*: For each unpacked rectangle, we pseudo-pack it horizontally and vertically at all corners to find all feasible positions (COAs). By "pseudo-pack" we mean the rectangle is packed temporarily and will be removed from the box in the future. Details for finding the feasible positions are omitted here, readers are referred to [15].

Fig. 8 illustrates the feasible positions of unpacked rectangles. For rectangle c, it can be packed horizontally and vertically at each of the corners c_1, c_2, c_3 and c_4. So, there are 8 feasible positions (i.e., 8 COAs) for rectangle c to pack. For rectangle d, similarly, there are also 8 feasible positions (8 COAs) to pack.

(3) *Pack a rectangle*: Under the current configuration, we enumerate all feasible COAs and calculate the caving degree for each of them. Then select the COA with the largest caving degree and pack the corresponding rectangle into the box in the corresponding position and orientation, until no rectangle remains outside the box or, none of the remaining rectangles can be packed in.

Note that, if there are multiple COAs with equal and largest caving degree then select a single COA with a deterministic way according to the following selecting order.

(a) Select the COA with the largest aspect ratio of the corresponding rectangle.
(b) Select the COA with the highest precedence of the bottom left of the corresponding rectangle.
(c) Select the COA with the smallest index of the corresponding rectangle.
(d) Select the COA with the corresponding rectangle packed horizontally, if both the horizontal and vertical packings are feasible.

Actually, above text describes a deterministic greedy packing algorithm. We call it basic algorithm A_0. On the basis of A_0, we introduce the backtracking process, and develop A_0 into a new algorithm A_1, i.e., algorithm *HRP*.

4. The algorithm

4.1. Basic program A_0

Greedy packing procedure (j)
Begin
Parameter j is the number of rectangles which have already been packed in.
Enumerate all feasible COAs under the current configuration.
While there exist some feasible COAs and $j < n$ ***do***
Calculate the caving degree for each COA.
Select COA with the largest caving degree.
If there are multiple COAs with the largest caving degree ***then***
Select a single COA according to the selecting order ((a) to (d) in Section 3.3 (3)).
Endif
Pack the corresponding rectangle into the box according to the selected COA and obtain a new configuration. Let $j \Leftarrow j + 1$.
Update the corners using the method described in Section 3.3(1).
Enumerate all feasible COAs under the current configuration.
Loop
Calculate the area usage of the box and return it.
End

4.2. Strengthened program A_1

Backtracking A_0
Begin
Input the width and height of the box and n rectangles R_i $(i = 1, 2, 3, \ldots, n)$. Let $k = 0$, where k is the number of rectangles which have already been packed in.
Enumerate all feasible COAs under the current configuration.
While there exist some feasible COAs and $k < n$ ***do***
Calculate the caving degree for each COA.
For each COA ***do***
According to the COA, pseudo-pack the corresponding rectangle and obtain a new configuration.
Let $j \Leftarrow k + 1$.
Call *Greedy packing procedure* (j) to pseudo-pack the remaining rectangles and get the area usage of the box.
If all n rectangles have already been packed in by *Greedy packing procedure* ***then***
Exit *while-loop* recurrence.

Else
Record the area usage of the box as the score of the corresponding COA.
Remove all pseudo-packed rectangles from the box.
Endif
EndFor
Select COA with the highest score.
If there are multiple COAs with the highest score ***then***
Select COA with the largest caving degree.
If there are multiple COAs with the largest caving degree ***then***
Select a single COA according to the selecting order ((a) to (d) in Section 3.3 (3)).
Endif
Endif
Pack the corresponding rectangle into the box according to the selected COA and obtain a new configuration. Let $k \Leftarrow k + 1$.
Update the corners using the method described in Section 3.3(1).
Enumerate all feasible COAs under the current configuration.
Loop
Output the total area of the packed rectangles, area usage and layout.
End

5. Experimental results

The basic algorithm A_0 and HRP are implemented by C#.net programming language running on an IBM notebook PC with 2.0 GHz processor and 256 MB memory. All 21 rectangle-packing instances provided by Hopper and Turton [2] are used in the experiments. For each instance, the box width is given and, the height needs to be determined. For each instance, the optimal height is known and the layout is perfect, that is, all rectangles are packed into the box without overlapping and the area usage of the box is 100%. For more details of these 21 instances, see Ref. [2].

Refs. [2,15] reflect the most advanced algorithms that have been published so far. Algorithm in [15] is based on the concept of flexibility. SA + BLF [2] means simulated annealing+ bottom left first and GA + BLF [2] means genetic algorithm+ bottom left first. Comparisons between SA+BLF, GA+BLF, the algorithm in [15], basic algorithm A_0 and HRP are shown in Tables 1 and 2. The runtime listed in the tables is that of the corresponding computer used. As an example, the layouts of instances 1, 4, 7, 10, 13 and 17 achieved by HRP are shown in Fig. 10.

For 21 rectangle-packing instances provided by Hopper and Turton [2], let the box height be equal to the optimal height. The area usage of the box of each instance is obtained by running HRP. 16 of the 21 instances achieved optimum solutions, i.e., all rectangles are packed into the box without overlapping and the area usage of the box is equal to 100%. 2 instances achieved optimum solutions in [15] (the computer is SUN Sparc20/71 with 71 MHz SuperSparc CPU and 64 MB RAM[3]). Table 1 shows the comparisons of the utilized area, the unutilized area and the runtime of 21 instances between the algorithm in [15], basic algorithm A_0 and HRP, where the runtime listed is the value of a single run. From Table 1, we can see that the runtime of some larger instances is less than that of some smaller instances. This is because for HRP the *Strengthened program* terminates successfully while all the remaining rectangles have been packed into the box after Calling *Greedy packing procedure*.

The algorithms mentioned in this paper can also be applied to the strip packing problem. For a given instance, if there are some rectangles which can not be packed into the current box, we may increase the box height and reapply the algorithm until all rectangles are packed into the box. For one algorithm and one instance we may use the so-called "relative distance"

$$100(\text{produced height} - \text{optimal height})/\text{optimal height}$$

[3] The computer used by [15] is about 30 times slower than the computer used by us.

Table 1
Comparisons of the utilized area, the unutilized area and the runtime between algorithm in [15], basic algorithm A_0 and HRP

Instance	# of rectangles	Box dimension (w x h)	Basic algorithm A_0			HRP			Algorithm in [15]		
			Utilized area	Unutilized area	Runtime (s)	Utilized area	Unutilized area	Runtime (s)	Utilized area	Unutilized area	Runtime (s)
1	16	20 × 20	384	16	0.01	400	0	0.05	392	8	1.48
2	17	20 × 20	386	14	0.01	400	0	0.23	392	8	2.42
3	16	20 × 20	384	16	0.01	400	0	1.12	390	10	2.63
4	25	40 × 15	585	15	0.02	600	0	0.08	596	4	13.35
5	25	40 × 15	594	6	0.02	600	0	0.10	600	0	10.88
6	25	40 × 15	588	12	0.02	600	0	0.28	600	0	7.92
7	28	60 × 30	1727	73	0.02	1800	0	2.58	1788	12	23.72
8	29	60 × 30	1756	44	0.01	1800	0	4.19	1785	15	34.02
9	28	60 × 30	1764	36	0.02	1800	0	2.50	1786	10	30.97
10	49	60 × 60	3510	90	0.05	3600	0	327.12	3565	35	438.18
11	49	60 × 60	3504	96	0.04	3600	0	36.59	3592	8	354.47
12	49	60 × 60	3544	56	0.04	3600	0	135.60	No report	No report	No report
13	73	60 × 90	5269	131	0.12	5400	0	55.44	5384	16	1417.52
14	73	60 × 90	5254	146	0.12	5400	0	29.17	5398	2	1507.52
15	73	60 × 90	5330	70	0.12	5400	0	51.13	5355	45	1466.15
16	97	80 × 120	9468	132	0.26	9586	14	873.38	9576	24	7005.73
17	97	80 × 120	9519	81	0.21	9600	0	327.61	9241	359	5537.88
18	97	80 × 120	9548	52	0.26	9594	6	577.59	9548	52	5604.70
19	196	160 × 240	37960	440	1.61	38308	92	4276.82	No report	No report	No report
20	197	160 × 240	38052	348	1.42	38355	45	3038.60	No report	No report	No report
21	196	160 × 240	37883	517	1.62	38337	63	3980.65	No report	No report	No report

Basic algorithm A_0 and HRP are run on IBM notebook PC with 2.0 GHz processor and 256 MB memory. The algorithm in [15] is run on SUN Sparc20/71 with 71 MHz SuperSparc CPU and 64 MB RAM.

Table 2
Comparisons of the relative distance and the runtime between SA+BLF, GA+BLF, basic algorithm A_0 and HRP

Instance	1, 2, 3	4, 5, 6	7, 8, 9	10, 11, 12	13, 14, 15	16, 17, 18	19, 20, 21
Basic algorithm A_0 (Relative distance, %/runtime, s)	6.67/0.01	6.67/0.02	4.44/0.03	2.22/0.05	1.48/0.10	1.39/0.25	0.97/1.88
HRP (Relative distance, %/runtime, s)	0/0.47	0/0.15	0/3.09	0/166.44	0/45.25	0.83/873.62	0.83/4276.86
SA+BLF (Relative distance, %/runtime, s)	4/42	6/144	5/240	3/1980	3/6900	3/22920	4/250860
GA+BLF (Relative distance, %/runtime, s)	4/60	7/120	5/180	3/780	4/2160	4/5160	5/46620

Basic algorithm A_0 and HRP are run on IBM notebook PC with 2.0 GHz processor and 256 MB memory. SA+BLF and GA+BLF are run on Pentium pro with a 200 MHz processor and 65 MB RAM.

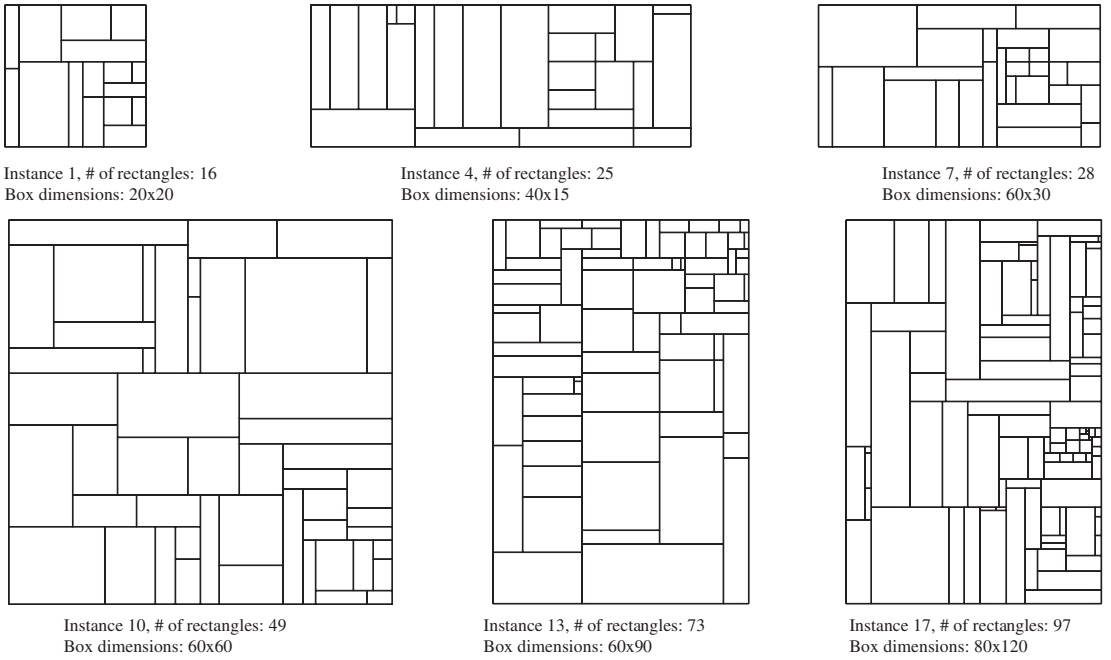

Fig. 10. The layouts of instances 1, 4, 7, 10, 13 and 17.

to measure the quality of the produced solution. Comparisons of the relative distance and the runtime between SA+BLF, GA+BLF,[4] basic algorithm A_0 and HRP are shown in Table 2. The relative distance and runtime listed are the average values of three instances.[5] For example, as shown in column 2 of Table 2, the relative distance and runtime are the average values of instances 1, 2 and 3.

[4] The computer used by SA+BLF and GA+BLF is about 10 times slower than the computer used by us.

[5] In literature, the former authors naturally chose three instances as a group. The scale of each instance in a group is almost the same.

6. Conclusions

A new heuristic rectangle packing algorithm is proposed based on the concept of caving degree of a corner-occupying action. High area usage is obtained within reasonable runtime. 16 of 21 instances provided by Hopper and Turton achieved optimum solutions with it. The experimental results demonstrate that this algorithm is rather efficient in solving the rectangle packing problem.

Acknowledgment

Thanks are due to the anonymous referees for many instructive suggestions. This work was partially supported by National Natural Science Foundation of China under Grant no. 10471051 and by NKBRPC (2004CB318000).

References

[1] Hopper E, Turton B. A genetic algorithm for a 2D industrial packing problem. Computers and Industrial Engineering 1999;37:375–8.
[2] Hopper E, Turton B. An empirical investigation of meta-heuristic and heuristic algorithm for a 2D packing problem. European Journal of Operational Research 2001;128:34–57.
[3] Guo PN, Cheng CK, Yoshimura T. An O-tree representation of non-slicing floorplans and its applications. In: DAC, 1999. p. 268–73.
[4] Pang Y, Cheng CK, Yoshimura T. An enhanced perturbing algorithm for floorplan design using the O-tree representation. ISPD 2000; 168–73.
[5] Chang YC, Chang Y-W, Wu GM, Wu SW. B*-tree: a new representation for non-slicing floorplans. In: DAC, 2000. p. 458–63.
[6] Murata H, Fujiyoshi K, Nakatake S, Kajitani Y. VLSI module placement based on rectangle-packing by the sequence pair. IEEE Transactions on Computer Aided Design 1996;15:1518–24.
[7] Tang X, Wong DF, FAST-SP: a fast algorithm for block placement based on sequence pair, ASP-DAC 2001; 521–6.
[8] Pisinger D. Denser placements in VLSI design obtained in O(n log log n) time. INFORMS Journal on Computing 2005, in press.
[9] Nakatake S, Murata H, Fujiyoshi H, Kajitani Y. Module placement on BSG-structure and IC layout application. In: Proceedings of International Conference on Computer Aided Design, 1996. p. 484–90.
[10] Hong XL, et al. Corner block list: an effective and efficient topological representation of non-slicing floorplan. In: IEEE, 2000, p. 8–12.
[11] Ma YC, et al. Stairway compaction using corner block list and its applications with rectilinear blocks. Proceedings of the 15th International Conference on VLSI Design 2002.
[12] Lin J-M, Chang Y-W. TCG: a transitive closure graph-based representation for non-slicing floorplans. In: DAC 2001. p. 764–9.
[13] Lin J-M, Chang Y-W. TCG-S: orthogonal coupling of P*-admissible representations for general floorplans. In: DAC 2002. p. 842–7.
[14] Lin J-M, Chang Y-W, Lin S-P. Corner sequence—a P-admissible floorplan representation with a worst case linear-time packing scheme. IEEE Trans on Very Large Scale Integration Systems 2003. p. 679–86.
[15] Wu Y-L, Huang WQ, Lau S, Wong C-K, Young GH. An effective quasi-human based heuristic for solving the rectangle packing problem. European Journal of Operational Research 2002;141:341–58.
[16] Huang WQ, Jin RC. The quasi-physical and quasi-sociological algorithm solar for solving SAT problem. Science in China, Series E (Technological Sciences) 1997;27(2):179–86.
[17] Huang WQ, Xu RC. Two personification strategies for solving circles packing problem. Science in China, Series E (Technological Sciences) 1999;42(6):595–602.
[18] Chen CS, Lee SM, Shen QS. An analytical model for the container loading problem. European Journal of Operational Research 1995;80: 68–76.
[19] Pisinger D, Sigurd MM. The two-dimensional bin packing problem with variable bin sizes and costs. Discrete Optimization 2005;2:154–67.
[20] Fekete SP, Schepers J. An exact algorithm for higher-dimensional orthogonal packing. Operations Research, in press.