A Hybrid Heuristic Algorithm for the Rectangular Packing Problem

Defu Zhang[1], Ansheng Deng[1], and Yan Kang[2]

[1] Department of Computer Science, Xiamen University, 361005, China
dfzhang@xmu.edu.cn
[2] School of Software, Yunnan University, Kunming, 650091, China

Abstract. A hybrid heuristic algorithm for the two-dimensional rectangular packing problem is presented. This algorithm is mainly based on divide-and-conquer and greedy strategies. The computational results on a class of benchmark problems have shown that the performance of the heuristic algorithm can outperform that of quasi-human heuristics.

1 Introduction

Packing problems have found many industrial applications, with different applications incorporating different constraints and objects. For example, in wood or glass industries, rectangular components have to be cut from large sheets of material. In warehousing contexts, goods have to be placed on shelves. In newspapers paging, articles and advertisements have to be arranged in pages. In the shipping industry, a batch of object of various sizes have to be shipped as many as possible in a larger container, a bunch of optical fibers have to be accommodated in a pipe with as small as possible. In VLSI floor planning, VLSI has to be laid. These applications can formalize as bin packing problems [1]. For more extensive and detailed descriptions of packing problems, the reader is referred to [1,2,3].

In this paper, two-dimensional rectangular packing problem is considered. This problem belongs to a subset of classical cutting and packing problems and has been shown to be NP hard [4,5]. Optimal algorithms for orthogonal two-dimension cutting are proposed in [6,7]. However, they might not be practical for large problems. Hybrid algorithms combining genetic with deterministic methods for the orthogonal packing problem are proposed [8, 9, 10]. An empirical investigation of meta-heuristic and heuristic algorithms of the orthogonal packing problem of rectangles is given by [11]. However, generally speaking, those non-deterministic algorithms are more time consuming and are less practical for problems having a large number of rectangles. Recently, an effective quasi-human heuristic, Less Flexibility First, for solving the rectangular packing problem is presented [12]. Namely, the rectangle with less flexibility should be packed earlier. This heuristic is fast and effective. Recently, several researchers have started to apply evolutionary algorithms to solve rectangular packing problem ([11, 13, 14, 15]. Based on the previous studies [16, 17], a rather fast and effective hybrid heuristic algorithm for the rectangular packing problem is presented.

V.S. Sunderam et al. (Eds.): ICCS 2005, LNCS 3514, pp. 783 – 791, 2005.

Computational results have shown that the performance of the hybrid heuristic algorithm can outperform that of quasi-human heuristics.

2 Mathematical Formulation of the Problem

Given a rectangular empty box and a set of rectangles with arbitrary sizes, we want to know if all rectangles can be packed into the empty box without overlapping. This problem can also be stated as follows.

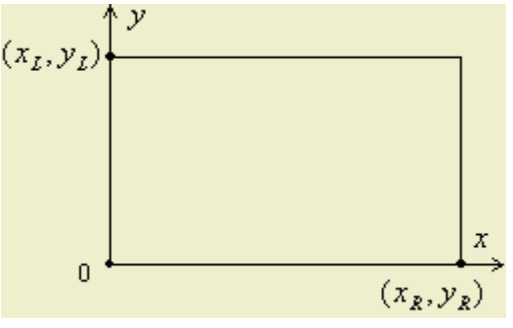

Fig. 1.

Given an empty box with length L and width W, and n rectangles with length l_i and width w_i, $1 \le i \le n$, take the origin of two dimensional Cartesian coordinate system at the left-down corner of the empty box, (x_L, y_L) denotes the left-up corner coordinates of the rectangular empty box and (x_R, y_R) denotes the right-down corner coordinates of this box (See Fig. 1.). Does there exist a solution composed of n sets of quadruples

$$P = \left\{ \left\langle (x_{li}, y_{li}), (x_{ri}, y_{ri}) \right\rangle \mid 1 \le i \le n, x_{li} < x_{ri}, y_{li} > y_{ri} \right\},$$

where, (x_{li}, y_{li}) denotes the left-up corner coordinates of rectangle i, and (x_{ri}, y_{ri}) denotes the right-down corner coordinates of rectangle i. For all $1 \le i \le n$, and the coordinates of rectangle i satisfies the following conditions:

1. $x_{ri} - x_{li} = l_i \wedge y_{li} - y_{ri} = w_i$ or $x_{ri} - x_{li} = w_i \wedge y_{li} - y_{ri} = l_i$.
2. for all $1 \le j \le n, j \ne i$, and rectangle i and j cannot overlap, namely, $x_{ri} \le x_{lj}$ or $x_{li} \ge x_{rj}$ or $y_{ri} \ge y_{lj}$ or $y_{li} \le y_{rj}$.
3. $x_L \le x_{li} \le x_R, x_L \le x_{ri} \le x_R$ and $y_R \le y_{li} \le y_L, y_R \le y_{ri} \le y_L$.

If there is no such a solution, then obtain a partial solution which minimizes the total area of the unpacked space.

It is noted that the packing process has to ensure the edges of each rectangle are parallel to the $x-$ and $y-$ axes respectively.

3 Hybrid Heuristic Algorithm

Many useful algorithms have recursive structure: to solve a given problem, they call themselves recursively one or more times to deal with closely related subproblems, so these algorithms are simple and effective. These algorithms typically follow a divide-and-conquer approach: they break the problem into several subproblems that are similar to the original problem but smaller in size, solve the subproblems recursively, and then combine these solutions to create a solution to the original problem.

The divide-and-conquer paradigm involves three steps at each level of the recursion [18]:

(1) Divide the problem into a number of subproblems.
(2) Conquer the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.
(3) Combine the solutions to the subproblems into the solution for the original problem.

Intuitively, we can construct a divide-and-conquer algorithm for the rectangular packing problem as follows:

(1) Pack a rectangle into the space to be packed. Divide the unpacked space into two subspaces (see Fig. 2).
(2) Pack the subspace by packing them recursively. If the subspace sizes are small enough to only pack a rectangle, however, just pack this rectangle into the subspace in a straightforward manner.
(3) Combine the solutions to the subproblems into the solution for the rectangular packing problem.

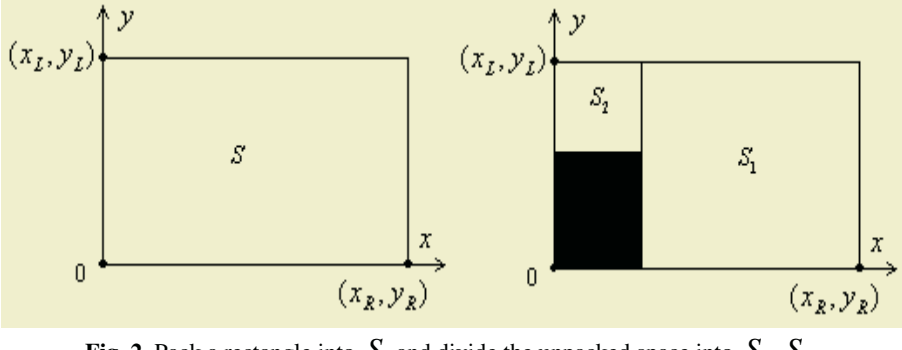

Fig. 2. Pack a rectangle into S and divide the unpacked space into S_1, S_2

It is noted that how to select a rectangle to be packed into S is very important to improve the performance of divide-and-conquer algorithm. In addition, two subspaces

S_1 and S_2, which to be first packed recursively also affect the performance of algorithm. In this paper, we present a greedy heuristic for selecting a rectangle to be packed, namely a rectangle with the maximum area or perimeter is given priority to pack. In detail, unpacked rectangles should be sorted by non-increasing order of area or perimeter size. The rectangle with maximum one should be selected to pack if it can be packed into the unpacked subspace.

Thus the divide-and-conquer procedure can be stated as follows (Fig.3):

```
Pack( S )
  if no rectangle can be packed into S
  then return;
  else
    Select a rectangle and pack it into S ;
    Divide the unpacked space into S1, S2 ;
    if the perimeter of S1 > the perimeter of S2
      Pack( S1 );
      Pack( S2 );
    else
      Pack( S2 );
      Pack( S1 );
```

Fig. 3. The divide-and-conquer procedure

For this recursive procedure, the average running time of the recursive procedure is $T(n) = \theta(n \lg n)$.

Due to the packing orders affect the packed ratio of the divide-and-conquer algorithm, several orders can be tried to enhance the packed ratio. If we strictly pack rectangles into S according to the order of area or perimeter, the way of this kind of packing is not correspond to practical packing in industry fields, so we can swap the orders of the big rectangle and small rectangle in order to keep the diversification of packing order. In detail, the greedy heuristic strategies are as follows:

1) Swap the order of two rectangles in given packing order;
2) Consider the computation in two kinds of area order and perimeter order by calling two times hybrid heuristic ().

The hybrid heuristic procedure can be stated as follows (see Fig. 4):

From the description of hybrid heuristic algorithm, we can know the average running time of this algorithm is $T(n) = \theta(n^3 \lg n)$. However, the worst running time of Heuristic1 [12] is $T(n) = \theta(n^5 \lg n)$, the worst running time of Heuristic2 [12]

is $T(n) = \theta(n^4 \lg n)$. Therefore, the computational speed of the hybrid heuristic algorithm is faster than that of Heuristic 1 and Heuristic 2.

```
Hybrid heuristic ()
  Repeat do
      for i=1 to n
        for j=i to n
          Swap the order of rectangle i and j in current order;
          Pack(S);
          Save the best order of rectangles and the best packed area so far;
         Current order =best order;
  Until (the best packed area has no improvement)
```

Fig. 4. The hybrid heuristic algorithm

4 Computational Results

Performance of the hybrid heuristic (HH) has been tested with seven different sized test instances ranging from 16 to 197 items [11]. These test instances have optimal solutions. The computational results are reported in Table 1. In order to understand HH more easily and compare HH with Heuristic 1 and Heuristic 2, we give their unpack ratio and running time comparison in Fig. 5, Fig. 6. Here, Heuristic 1 and Heuristic 2 are not implemented in this paper, they are run on a SUN Sparc20/71, a machine with a 71 MHz SuperSparc [12], so the % of unpacked area and running time are directly taken from [12]. Our experiments are run on a Dell GX260 with a 2.4GHz CPU. It is noted that the computational results of C7 are not reported in [12], and so is Heuristic 1 for C43. So in Fig. 5 and Fig. 6, % of unpacked area and the running time for Heuristic 1 are shown to be empty. In addition, we give four packed results on test instances C1 and C73 for HH in Fig. 7 and Fig. 8 respectively.

On this test set, as shown in Table 1, HH runs much faster for all test instances than Heuristic1 and Heuristic 2. For C1-C6, the average running time of HH is less than 1.5s, for C7, the average running time of HH is less than 13.47s. It has shown that the actual running time of HH accords with its computational complexity. The unpacked area for HH ranges from 0% to 3.5% with the average unpacked ratio of 0.88%. The average unpacked ratio of Heuristic 1 and Heuristic 2 is 0.92 and 3.65 respectively. The average unpacked ratio of HH is lower than that of Heuristic 1 and Heuristic 2. From Fig. 5 and Table 1, we can observe that the packed density increases with the increasing of the number of rectangles. The running time of Heuristic 1 increases more badly. With the increasing of the number of rectangles, it is imaginable that the computational speed of Heuristic 1 for practical applications is unacceptable.

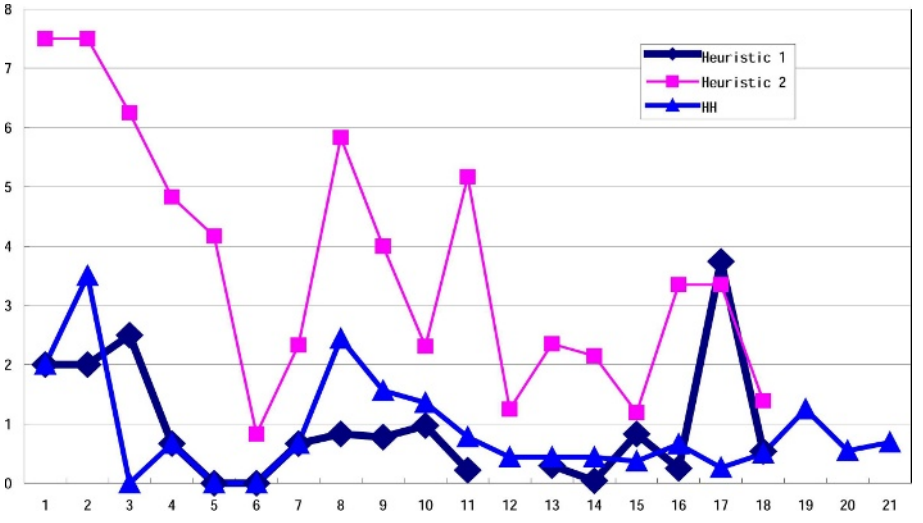

Fig. 5. % of unpacked area for Heuristic 1, Heuristic 2 and HH

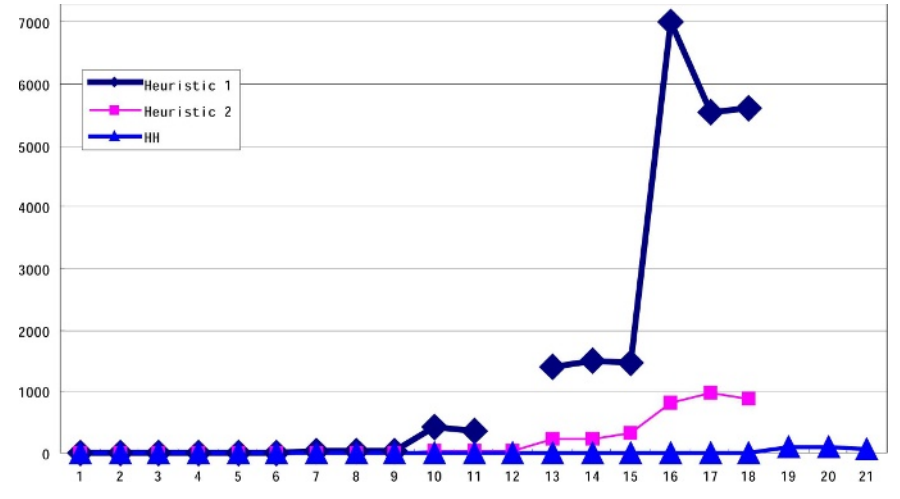

Fig. 6. CPU time comparison of Heuristic 1, Heuristic 2 and HH

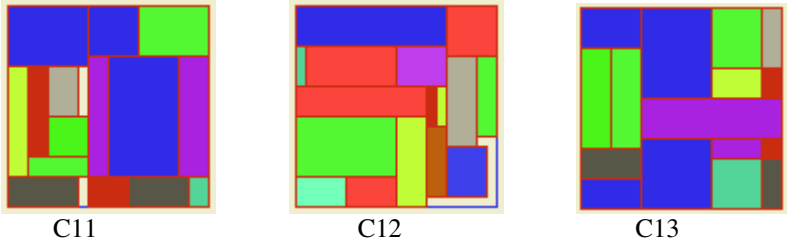

Fig. 7. Packed results of C1 for HH

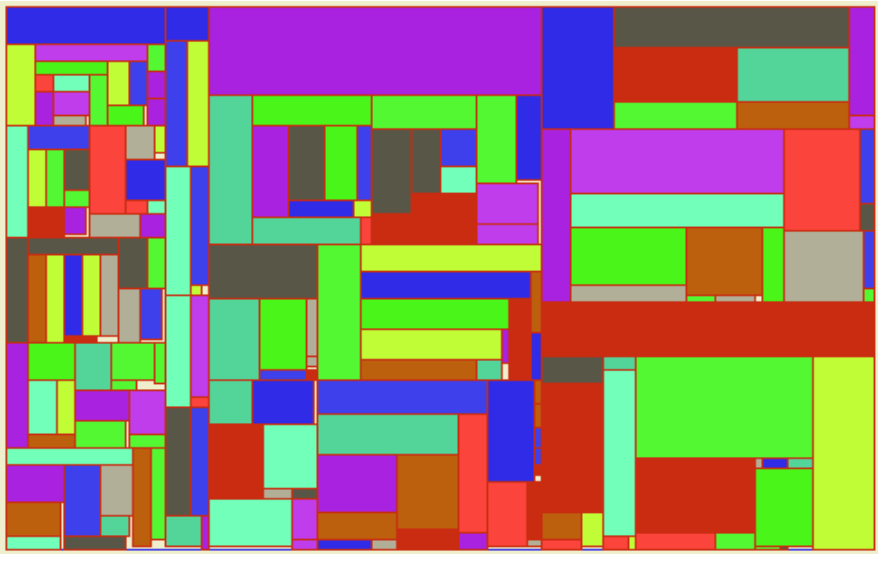

Fig. 8. Packed result of C73 for HH

Table 1. Experimental results on Heuristic 1, Heuristic 2 and HH

Test instances		n	L×W	CPU time (seconds)			% of unpacked area		
				Heuristic1	Heuristic2	HH	Heuristic1	Heuristic2	HH
C1	C11	16	20×20	1.48	0.10	0.0	2.00	7.50	2
	C12	17	20×20	2.42	0.18	0.0	2.00	7.50	3.5
	C13	16	20×20	2.63	0.17	0.0	2.50	6.25	0
C2	C21	25	15×40	13.35	0.80	0.05	0.67	4.83	0.67
	C22	25	15×40	10.88	0.98	0.05	0.00	4.17	0.0
	C23	25	15×40	7.92	7.92	0.0	0.00	0.83	0.0
C3	C31	28	30×60	23.72	2.07	0.05	0.67	2.33	0.67
	C32	29	30×60	34.02	1.98	0.05	0.83	5.83	2.44
	C33	28	30×60	30.97	2.70	0.05	0.78	4.00	1.56
C4	C41	49	60×60	438.18	32.90	0.44	0.97	2.31	1.36
	C42	49	60×60	354.47	33.68	0.44	0.22	5.17	0.78
	C43	49	60×60		29.37	0.33		1.25	0.44
C5	C51	73	90×60	1417.52	234.52	1.54	0.30	2.35	0.44
	C52	73	90×60	1507.52	237.58	1.81	0.04	2.15	0.44
	C53	73	90×60	1466.15	315.60	2.25	0.83	1.19	0.37
C6	C61	97	120×80	7005.73	813.45	5.16	0.25	3.35	0.66
	C62	97	120×80	5537.88	975.30	5.33	3.74	3.35	0.26
	C63	97	120×80	5604.70	892.72	5.6	0.54	1.39	0.5
C7	C71	196	240×160			94.62			1.25
	C72	197	240×160			87.25			0.55
	C73	196	240×160			78.02			0.69
Average time and % of unpacked area				1379.97	199.00	13.47	0.92	3.65	0.88

5 Conclusions

A novel hybrid heuristic algorithm for the rectangular packing is presented in this paper. This algorithm is very simple and intuitional, and can fast solve the rectangular packing problem. The computational results have shown that HH outperform heuristic 1 and Heuristic 2 in two ways of % of unpacked area and the CPU time. So HH may be of great practical value to the rational layout of the rectangular objects in the engineering fields, such as the wood-, glass- and paper industry, and the ship building industry, textile and leather industry. Further improve the performance of hybrid heuristic algorithm and extend this algorithm for three-dimensional rectangular packing problems are the future work.

References

1. Andrea Lodi, Silvano Martello, Michele Monaci. Two-dimensional packing problems: A survey. European Journal of Operational Research 141 (2002) 241–252
2. K.A. Dowaland, W.B. Dowsland. Packing problems. European Journal of Operational Research 56 (1992) 2–14
3. David Pisinger. Heuristics for the container loading problem. European Journal of Operational Research 141 (2002) 382–392
4. D.S. Hochbaum Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and VLSI. Journal of the Association for Computing Machinery 32 (1) (1985) 130–136
5. J. Leung, T. Tam, C.S. Wong, Gilbert Young, Francis Chin. Packing squares into square. Journal of Parallel and Distributed Computing 10 (1990) 271–275
6. J.E. Beasley. An exact two-dimensional non-guillotine cutting tree search procedure. Operations Research 33 (1985) 49–64
7. E. Hadjiconstantinou, N. Christofides. An optimal algorithm for general orthogonal 2-D cutting problems. Technical report MS-91/2, Imperial College, London, UK
8. S. Jakobs. On genetic algorithms for the packing of polygons. European Journal of Operational Research 88 (1996) 165–181
9. D. Liu, H. Teng. An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles. European Journal of Operational Research 112 (1999) 413–419
10. C.H. Dagli, P. Poshyanonda. New approaches to nesting rectangular patterns. Journal of Intelligent Manufacturing 8 (1997) 177–190
11. E. Hopper, B.C.H. Turton. An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. European Journal of Operational Research 128 (2001) 34–57
12. Yu-Liang Wu, Wenqi Huang, Siu-chung Lau, C.K. Wong, Gilbert H. Young. An effective quasi-human based heuristic for solving the rectangle packing problem. European Journal of Operational Research 141 (2002) 341–358
13. Andrea Lodi, Silvano Martello,and Daniele Vigo. Heuristic and Metaheuristic Approaches for a Class of Two-Dimensional Bin Packing Problems, INFORMS Journal on Computing 11 (1999) 345–357
14. E. Hopper. Two-Dimensional Packing Utilising Evolutionary Algorithms and other Meta-Heuristic Methods, PhD Thesis, Cardiff University, UK. 2000

15. J. Puchinger and G. R. Raidl. An evolutionary algorithm for column generation in integer programming: an effective approach for 2D bin packing. In X. Yao et. al, editor, Parallel Problem Solving from Nature - PPSN VIII, volume 3242, pages 642–651. Springer, 2004
16. De-fu Zhang, An-Sheng Deng. An effective hybrid algorithm for the problem of packing circles into a larger containing circle. Computers & Operations Research 32(8) (2005) 1941–1951
17. Defu Zhang, Wenqi Huang. A Simulated Annealing Algorithm for the Circles Packing Problem. Lecture Notes in Computer Science (ICCS 2004) 3036 (2004) 206–214
18. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. Introduction to Algorithms. Second Edition, The MIT Press (2001)