A Genetic Algorithm for the Two-dimensional Single Large Object Placement Problem

László Pál

Sapientia University, Department of Mathematics and Informatics
Libertatii 1, RO-530104 Miercurea-Ciuc, Romania
E-mail: pallaszlo@sapientia.siculorum.ro

Abstract: The two-dimensional Single Large Object Placement Problem (SLOPP) problem consists of determining a cutting pattern of a set of n small rectangular piece types (little object) on a rectangular stock plate (large object) of length L and width W, as to maximize the sum of the profits of the pieces to be cut. Each piece type i, i = 1, . . ., m, is characterized by a length li, a width wi, a profit (or weight) ci and an upper demand value bi. Only guillotine cuts are allowed and the pieces may be rotated by 90°. In this paper three heuristic algorithm were applied to constrained two-dimensional cutting problems and the results were compared.

Keywords: cutting and packing, genetic algorithm

1 Introduction

Cutting and packing problems are a special case of combinatorial optimization problems [7]. They are encountered in numerous realworld applications such as computer science, industrial engineering, logistics, manufacturing, etc.

According to the new improved topology of Cutting and Packing problems [17], our problem falls into the two-dimensional, rectangular SLOPP (Single Large Object Placement Problem) category.

The two-dimensional, rectangular SLOPP defines a problem category in which a weakly heterogeneous assortment of small items has to be assigned to a given, limited set of large objects. The set of large objects is not sufficient to accommodate all the small items. The value or the total size of the accommodated small objects has to be maximised, or, alternatively, the corresponding waste has to be minimised. In this paper only guillotine cuts are allowed. An instance of the two-dimensional, rectangular and guillotine SLOPP problem consists of a large stock rectangle of given dimensions L X W and m types of smaller rectangles (pieces) where the ith type has dimensions $l_i \times w_i$. Furthermore, each type i, $i=$

1,…,m, is associated with a profit c_i and an upper bound b_i. The problem is now to cut off from the large rectangle a set of small rectangles such that:

a) All pieces have fixed *orientation,* i.e. a piece of length l and width w is different from a piece of length w and width l (when $l \neq w$).

b) All applied cuts are of *guillotine* type, i.e. they must run from end to end on the rectangle being cut.

c) There are at most b_i rectangles of type i (the demand constraint of the *i*th piece) in the (feasible) cutting pattern.

d) The overall profit obtained $\sum_{i=1}^{m} c_i x_i$ where x_i denotes the number of rectangles of type i in the cutting pattern, is *maximized.*

In this paper we consider the unweighted version of the two-dimensional, rectangular SLOPP in which the profit c_i of the *i*th piece is exactly its area and the pieces may be rotated by 90°. We wish to find a cutting pattern that minimizes the unused area (trim loss).

Many authors have considered guillotine cutting problems. The unconstrained guillotine SLOPP problem has been solved optimally by dynamic programming [8, 9]. For the constrained guillotine SLOPP problem, Christofides and Whitlock [4] have developed a tree search procedure based upon a depth-first search method, and Viswanathan and Bagchi [18] have also used a branch and bound method based upon a best-first search method for solving exactly the problem. Hifi [10] improved Viswanathan and Bagchi's [18] exact algorithm for the constrained guillotine two-dimensional cutting problem. Hifi [12] give several approximate algorithms for the constrained two-dimensional cutting problem based upon a strip generation procedure.

This paper is organised as follows: In Section 2 we present two decoding heuristics while in Section 3 we describe the genetic algorithm and its attributes. In Section 4 we present the experimental results according to the decoding heuristics.

2 Decoding Heuristics

A packing pattern may be represented by a permutation, which corresponds to the sequence in which the small rectangles are packed. Other representations may be found in [2, 6]. In this paper we concentrate on the permutation representation. Now if we have a permutation, we know the order of packing the small pieces, but we may still consider different strategies for packing the pieces. I have developed a placement strategy using the Wang's idea [16] denoted by WDH. This process is also called decoding, and the pattern thus formed is also called a phenotype.

2.1 The WDH Placement Algorithm

The heuristic it is based on the observation that all guillotine cutting patterns can be obtained by means of horizontal and vertical builds (Fig. 1). Waste arises if there is a mismatch in the relevant dimension of the rectangles being combined. Demanded rectangles and rectangles resulting from horizontal and vertical builds are called guillotine rectangles. The area wasted inside a guillotine rectangle R is its internal waste. The total waste of R is the waste which will result if R is "placed" in the stock rectangle (Fig. 2).

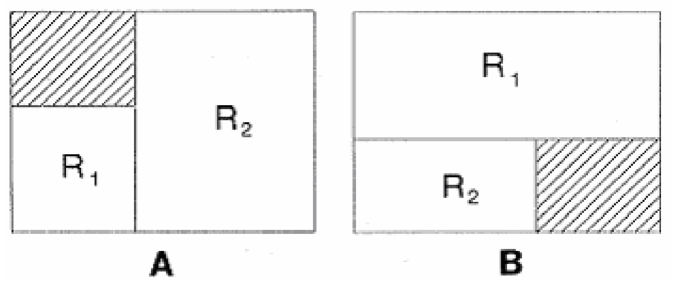

Figure 1
Horizontal (a) and vertical (b) builds

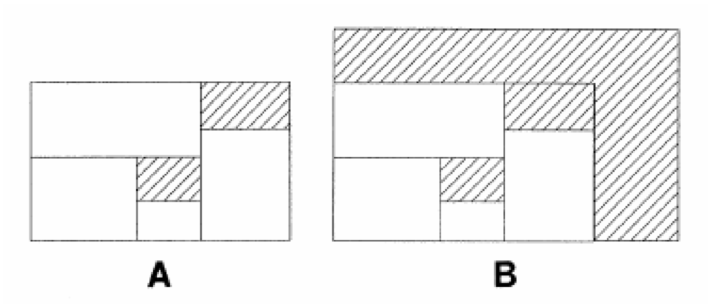

Figure 2
Internal (a) and total (b) waste

To reducing the internal waste we have developed two variants of the placement heuristic: WDH1 and WDH2. The main idea is common in both of the algorithms; only the internal waste filling procedure is different.

The main idea: in every step we create a guillotine rectangle and try to fill the gap to reducing the internal waste. This placement technique is summarized in the following steps:

R_i, i=1..n modules (rectangles) to be placed

P_i, i=1..2 the top-left and bottom-right corners of the guillotine rectangle

$$n = \sum_{i=1}^{m} b_i$$

newOBJ – represents the aria of the new guillotine rectangle

```
1. Place module 1 at the bottom-left corner of the sheet
2. Set the guillotine rectangle module 1
3. Set Pi, i=1..2
4. For i=1 to n
        Set OBJ to a big value
        For j=1 to 2
            Place the bottom-left corner of Ri on Pj
            Check boundary conditions
            IF conditions satisfied THEN Calculate the newOBJ
            IF newOBJ is less than OBJ THEN
                OBJ = newOBJ
                Save placement of module Ri
            EndIf
        EndFor
        Set the new guillotine rectangle
        Set the new Pi of guillotine rectangle
        If there is gap then we fill it using a Greedy algorithm
EndFor
```

In the next steps we describe the two “gap-fill” algorithms. We consider the gap as a strip. In the first case the reminded rectangles are sorted into height or width order. The gap will be filled horizontally or vertically by iterating over the sorted rectangles (Fig. 3). In the second case the gap will be filled by iterating simply over the reminded rectangles (Fig. 4).

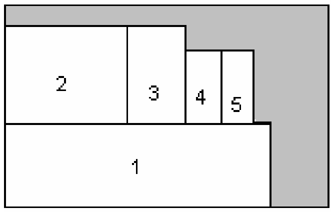

Figure 3
WDH1 heuristic

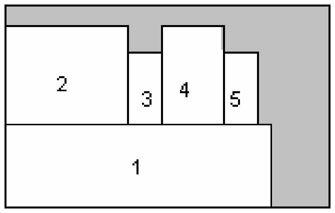

Figure 4
WDH2 heuristic

2.2 Improved First Fit Heuristic

The FF heuristic is a well-known algorithm and was considered by many researchers [3, 15]. In this case the rectangles are first sorted into height order, with the orientation being chosen so as to maximise height. The first rectangle is placed in the upper left corner of the sheet. Subsequent rectangles are placed to the right of this until there is insufficient space to add another. The first row is now complete, with a height equal to that of the first rectangle. This process is repeated until no more rows can fit on the sheet (see Fig. 5).

We improve the First Fit heuristic by fitting multiple rectangles into the same column. Only rectangles of the same width (or less) can be placed in the same column. When we have completed a row, we do not immediately proceed to the next. Instead, we iterate over all remaining unplaced rectangles, looking for any which have the same or less width as one of the columns, and are sufficiently small to be placed in the same column without increasing the height of the row (see Fig. 6).

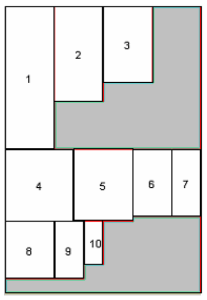

Figure 5
FF heuristic

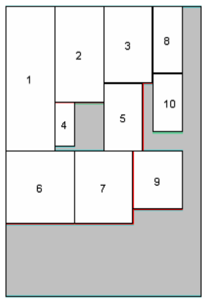

Figure 6
IFF heuristic

3 Genetic Algorithm

A genetic search algorithm is a heuristic search process that resembles natural selection. There are many variations and refinements, but any genetic algorithm has the features of reproduction, crossover and mutation. Initially a population is selected, and by means of crossovers among members of the population or mutation of members, the better of the population will remain.

3.1 Representation and Decoding

We use a permutation π representation for the chromosome:

$$\pi = \pi(1), \pi(2), ..., \pi(n) \text{ ([13])}$$

Here $\pi(i)$, $i = 1, 2, ..., \mathrm{n}$ denotes the index of the rectangles and $n = \sum_{i=1}^{m} b_i$.

In the chromosome we also include an additional entry for the rotated version of each rectangle. We suggest that $\pi(i)$ has a sign to indicate how the rectangle $\pi(i)$ is packed [14]. The positive denotes that the rectangle is packed by the long edge parallel to the x axes. The negative denotes that the rectangle is packed by the short edge parallel to the x-axes, i.e. the rectangle is rotated by 90°. The advantage of this data structure is it is easy to create new permutations by changing the sequence which is done by the crossover and mutation operators of GAs.

For decoding this genotype we use the WDH and IFF heuristics described above. Instead of sorting rectangles by height order, we sort them in order according to the chromosome. Thus the chromosome is a queue of rectangles to be placed by the heuristic.

3.2 Recombination and Mutation

The crossover operator creates one or more offspring solutions by combining two parents. For each pair of strings (parents), two cut points are generated randomly along the positions of the strings. The elements lying between these two cut points of the first string are taken out. The vacant positions are filled with respect to the same order by elements, which also appear between the cut points on the first string. This gives the first child. Now certain elements of the second string are removed. They are replaced by elements of the first string between the cut points, with respect also to the same order of appearance.

The mutation operator used in this paper consists of two parts. One part exchanges two random elements or blocks of the every new permutation at a small mutation rate. Another part reverses the sign of every element of the new permutations, i.e. rotates every rectangle by 90°, with a low probability.

One additional crossover operator was specifically designed for the permutation representation. In the crossover operator two crossover points are randomly chosen. The crossover region defined by the two crossover points is transmitted directly from the first parent to the offspring. All remaining positions are filled with the remaining elements of the first parent in the order given by the second parent.

3.3 Objective Function

In this paper I consider only one objective which is the wastage minimization. Our objective function is given by the total area of all little objects divided by the large object area:

$$Fitness = \frac{\sum_{j=1}^{n} l_j w_j}{LW}$$

where

– n is the number of the piece cut

– $l_j w_j$ is the area of piece j

– LW is the total area of the sheet

4 Experimental Results

I conducted my study on 15 small instances extracted from [11]. The number of little objects (rectangles) in every instance ranges from 20 to 60. These benchmark problems, from the literature, can be downloaded from http://www.laria.u-picardie.fr/hifi/OR-Benchmark/2Dcutting/2Dcutting.html.

My objective was to compare the performance of the three algorithm according to the wastage minimization. In this paper I didn't consider the run time studying. I coded the the heuristic methods in Java and ran them on a PC (Pentium IV, 500 MB RAM. For each test problem, all the heuristics were run 15 times and I took averages of the result. The quality of a solution is measured by the trim-loss percentage (wastage percentage).

There is no clear evidence that larger populations will produce better offspring. I tried populations of sizes from 20 to 70, and I found that the best results occurred when the sizes of the populations are about 50. Each run was terminated after the 100 iterations, when the trim loss remained stable. For every decoding methods I used the two crossover operator and the two mutation operator. In terms of choices of mutation rates I found that lower mutation rates usually work better. In both of the them the mutation operator were 10%.

The results for the instances from [11] are presented in Table 1. It provides more detailed results on each of the 15 instances for WDH1, WDH2 and IFF. The W_{avg} (average waste percent) column list average solution values over 15 runs per instance, W_{min} (average minimum waste) and W_{max} (average maximum waste) show solution values of the best respectively the worst run. The second column from Table 1 list the different demanded rectangles (m) while the third column list the total number of the demanded rectangles (n).

Table 1

Detailed results of selected algorithms

				GA+WDH1			**GA+WDH2**			**GA+IFF**		
Nr.	Inst.	m	n	W_{avg}	W_{min}	W_{max}	W_{avg}	W_{min}	W_{max}	W_{avg}	W_{min}	W_{max}
1	2s	10	23	**0,030**	0,040	0,018	**0,032**	0,050	0,015	**0,031**	0,062	0,017
2	3s	20	62	**0,020**	0,028	0,016	**0,022**	0,037	0,010	**0,033**	0,053	0,021
3	A1s	20	62	**0,014**	0,028	0,005	**0,016**	0,030	0,005	**0,015**	0,024	0,005
4	A2s	20	53	**0,027**	0,034	0,006	**0,028**	0,033	0,006	**0,030**	0,061	0,006
5	A3	20	46	**0,030**	0,046	0,015	**0,032**	0,048	0,013	**0,031**	0,057	0,018
6	A4	20	35	**0,030**	0,052	0,022	**0,052**	0,069	0,029	**0,043**	0,034	0,032
7	A5	20	45	**0,030**	0,060	0,016	**0,035**	0,058	0,016	**0,045**	0,087	0,016
8	CHL1s	30	63	**0,018**	0,030	0,003	**0,030**	0,056	0,006	**0,029**	0,043	0,009
9	CHL2s	10	19	**0,032**	0,062	0,028	**0,035**	0,048	0,028	**0,048**	0,066	0,039
10	CHL6	30	65	**0,019**	0,041	0,005	**0,021**	0,051	0,008	**0,019**	0,039	0,004
11	Hchl3s	10	51	**0,036**	0,052	0,019	**0,040**	0,056	0,023	**0,036**	0,049	0,026
12	Hchl4s	10	32	**0,040**	0,057	0,021	**0,049**	0,072	0,020	**0,062**	0,093	0,036
13	Hchl6s	22	60	**0,042**	0,056	0,019	**0,060**	0,099	0,041	**0,058**	0,071	0,048
14	OF1	10	23	**0,040**	0,053	0,044	**0,053**	0,077	0,045	**0,043**	0,051	0,031
15	OF2	10	24	**0,045**	0,076	0,026	**0,052**	0,066	0,026	**0,076**	0,086	0,055

These results show that with WDH2 and IFF methods we can achieve similar results. It can be observed that average solution qualities do in general not differ very much. On the other hand it is clear that the WDH1 method produced better results than the WDH and IFF heuristics. In Fig. 7, the mean trim losses of the population in a particular run are plotted against iteration number.

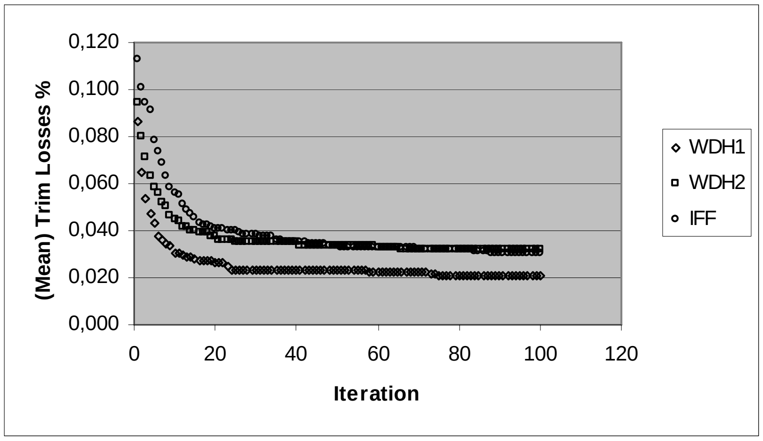

Figure 7
Mean trim losses in a population versus iteration numbers for instance 11

Generally I found that the use of these heuristics produce reasonable results for this problem class. Of all the test problems the trim loss goes from 0% to 5% which are within acceptable standards.

An example for cutting patterns obtained by WDH1, WDH2 and IFF for instance 11 is shown in Fig. 8.

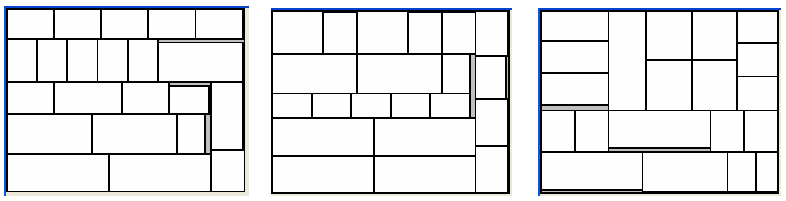

Figure 8
Cutting patterns for instance number 11 obtained by WDH1, WDH2 and IFF heuristics

Conclusions

I considered different heuristic algorithms for two-dimensional, rectangular and guillotine SLOPP problem. I compared the WDH1, WDH2 and IFF methods and found that with WDH2 and IFF we can get similar results while the WDH1 outperforms them according to the wastage minimization.

In further works I will use different chromosome representation to extend these heuristic algorithms for larger instances.

References

[1] J. E. Beasley: Algorithms for unconstrained two-dimensional guillotine cutting. Journal of the Operational Research Society, 1985, 36, 297-306

[2] J. E. Beasley: An exact two-dimensional nonguillotine cutting tree search procedure, Operations Research 33/1 (1985) 49-64

[3] J. O. Berkey, P. Y. Wang: Two-dimensional finite bin packing algorithms. Journal of the Operational Research Society, 38:423-429, 1987

[4] N. Christofides, C. Whitlock: An algorithm for two-dimensional cutting problems, Operations Research 25 (1977) 3 1-44

[5] K. A. Dowsland, W. Dowsland: Packing problems, European Journal of Operational Research 56 (1992) 2-14

[6] K. A. Dowsland: Genetic algorithms – a tool for OR, Journal of the Operational Research Society 47 (1996) 550-561

[7] H. Dyckhoff, A typology of cutting and packing problems, European Journal of Operational Research 44 (1990) 145-159

[8] P. C. Gilmore, R. E. Gomory, Multistage cutting problems of two and more dimensions, Operations Research 13 (1965) 94-119

[9] P. C. Gilmore, R. E. Gomory, The theory and computation of knapsack functions, Operations Research 14 (1966) 1045-1074

[10] M. Hifi.: An improvement of Viswanathan and Bagchi’s exact algorithm for constrained two-dimensional cutting stock, Computers and Operations Research 24 (1997) 727-736

[11] M. Hifi., C. Roucairol: Approximate and exact algorithms for constrained (un)weighted two-dimensional two-staged cutting stock problems, Journal of Combinatorial Optimization 5 (2001) 465-494

[12] M. Hifi., R. M’Hallah: Strip generation algorithms for constrained two-dimensional two-staged cutting problems, Euro. J. of Op. Res., ?? (2004) Article in Press

[13] S. Jakobs: On genetic algorithms for packing polygons, Euro. J. of Op. Res., 88 (1996) 165-181

[14] D. Liu, H. Teng: An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles, Euro. J. of Op. Res., 112 (1999) 413-420

[15] J. Puchinger, G. R. Raidl, and G. Koller. Solving a real-world glass cutting problem. In J. Gottlieb and G. R. Raidl, editors, Evolutionary Computation in Combinatorial Optimization – EvoCOP 2004, Volume 3004 of LNCS, pages 162-173. Springer, 2004

[16] P. Y. Wang,: Two algorithms for constrained two-dimensional cutting stock problems, Operations Research 31/3 (1983) 573-586

[17] G. Wascher, H. Haubner, H. Schumann: An Improved Typology of Cutting and Packing Problems, Working Paper No. 24, Last Revision: 2006-01-16, Faculty of Economics and Management Magdeburg

[18] K. V. Viswanathan, A. Bagchi, A best first search method for constrained two-dimensional cutting stock problems, Operations Research 41 (1993) 768-776