UNIVERSIDADE DE SÃO PAULO
Campus de São Carlos
INSTITUTO DE CIÊNCIAS MATEMÁTICA DE SÃO CARLOS

na and-or-graph representation for two-dimensional cutting problems

MORABITO,R.N.; ARENALES ,M.N.;
ARCARO, V.F.

N° 60

NOTAS do ICMSC-USP

ISSN 0103-2577

An and-or-graph representation for two-dimensional cutting problems

MORABITO, R.N.; ARENALES, M.N.;
ARCARO, V.F.

nº 60

São Carlos (SP)
1990

An And-Or-Graph Representation for Two-Dimensional Cutting Problems

R. N. MORABITO, M. N. ARENALES
Depto. de Ciências de Computação e Estatística - ICMSC-USP
13.560 - São Carlos - SP - Brasil. c.p. 668

and

V. F. ARCARO
Faculdade de Engenharia Civil - UNICAMP
13.100 - Campinas - SP - Brasil

Abstract: The problem of generating cutting patterns for a rectangular plate is studied and proposed an and-or-graph representation to the problem. To search the graph we used a combination of two strategies: depth-first and hill-climbing. Additionally some heuristics are considered and computational results are presented.

Keywords: Cutting Problem, Nesting, And-Or-Graph

1. Introduction

The two-dimensional cutting problem consists of cutting a

rectangular plate of dimensions (W,L) into smaller rectangular pieces of dimensions (w_i, l_i), which have utility values π_i, $i=1,\ldots,m$, in such a way to maximize the sum of the utility values of the produced pieces.

A set of smaller pieces defines a *cutting pattern* if these pieces can be produced by a sequence of possible cuts on (W,L). All the produced pieces that differ from (w_i, l_i), $i=1,\ldots,m$, are considered as waste.

Let a_i be the number of produced pieces (w_i, l_i). Then the problem can be stated as:

$$\begin{aligned} &max\ g = \pi_1 a_1 + \ldots + \pi_m a_m \\ &\text{such that } (a_1,\ldots,a_m)^T \qquad (1) \\ &\text{corresponds to a cutting pattern,} \end{aligned}$$

If a demand of (w_i, l_i) pieces must be satisfied, and possibly several (W,L) plates have to be cut (in this case, the problem is refered as cutting stock problem), the problem (1) is used to generate columns to a linear program (see Gilmore and Gomory [1965]). In this case the utility values are functions of the simplex multipliers.

An additional constraint on the cuts will be imposed to solve the problem (1): each cut produces just two rectangles. These cuts are called *guillotine cuts*.

The cuts are made, at a first *stage*, parallel to an edge of the rectangle and then in a subsequent stage, perpendicular to the previous cuts and so on. If there exists an upper bound to the number of stages then the cuts are called *staged*, otherwise called *non-staged*. The figure 1 shows a 8-*stage* guillotine cutting pattern.

To simplify the illustration, only one cut has been made at each stage and only one produced rectangle has been cut.

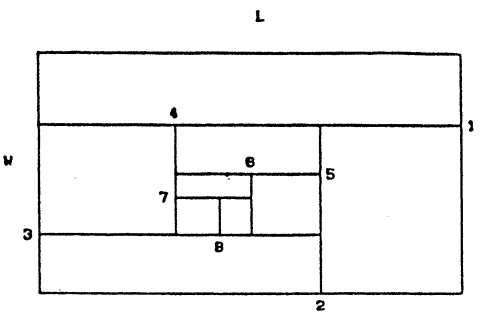

FIGURE 1 - 8-STAGE GUILLOTINE CUTTING PATTERN

In this paper we focus on the problem (1) with guillotine and non-staged cuts.

The dynamic programming has been used to solve the problem. Beasley [1985] compiled the main works and showed how the discretization described by Herz [1972] can be used to improve the performance of the recursive formulas and used it to provide a heuristics for large problems.

A solution via graph search was introduced by Herz, who noted that the feasible solutions space can be, without loss of optimality, reduced quite a lot if it was imposed rules to avoid duplications of patterns or unnecessary cuts. These rules were discussed in the earlier paper (Morabito et al [1989]).

Here we formally describe an and-or-graph representation to generate all the feasible cutting patterns (Herz and Christofides-Whitlock [1977] had implicity used this representation). Through the graph representation it is easy to see the Herz's recursive procedure as a classical *depth-first* search.

We propose a *hybrid search*, combining *depth-first* search and *hill-climbing* strategy. We also propose some heuristics to avoid cuts with lower possibility of generating the optimal cutting pattern. Additionally we present computational results to randomly generated examples and to the examples given by Herz and Beasley.

2. The And-Or-Graph Representation

A search process can be considered as the traversal of a directed graph in which each node represents a subproblem (state) and each arc represents a relationship between the nodes. It is possible to define a problem as a search in the graph by specifying the initial node, the final nodes and the rules that define the legal moves.

In the two-dimensional cutting problem the rectangular plate is represented by the initial node and the small rectangles are represented by the final nodes. The rules that define the legal moves are the possible cuts on a rectangle (a guillotine cut, for example. Other rules that avoid duplication of patterns can be formulated). The possibility of no cut is also taken as a rule.

Therefore, a subset of cuts can be applied to the rectangular plate (A in figure 2) corresponding to the branching degree of the initial node . The two succeeding rectangles (B,C and D,E in figure 2) after each cut, correspond to the intermediary nodes. Other subset of cuts can be applied to an intermediary rectangle and so on until the given rectangular pieces be found as the final nodes. The representation of the above cutting process is illustrated in figure 2.

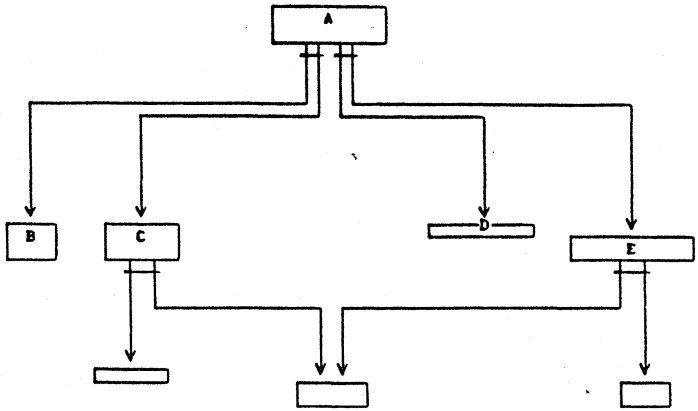

FIGURE 2 - AND-OR-GRAPH

An *and-or-graph* is the type of structure used to represent a problem which is broken into a set of smaller problems that are then solved. This decomposition (or reduction) generates arcs, called *and-arcs*, which in our case are pairs, taking the current node (rectangle) into new nodes (rectangles produced by a cut) which, although are treated as individual nodes, none of these new nodes alone can constitute a complete solution to the previous node. On the other hand, at each node, a number of *or-arcs* can also occur denoting alternative approaches in handling the rectangle from which they emanate.

A *complete path* in the above graph represents a sequence of cuts from the initial rectangular plate which ends in one or more rectangular pieces, and then it provides a cutting pattern, that is, a feasible solution to the problem (1). The path to be chosen is the one that maximizes the value of g (the objective function of problem (1)) and therefore represents the best cutting pattern on the rectangular plate.

The and-or-graph to be searched could, in principle, be completely constructed from the rules that define legal moves. This construction is not always necessary since, in practice, the graph can be very large and only some parts of it need to be

examined. For this reason, it is better to define the graph implicitly, and let the search procedure to decide at any given time which operation is to be applied to the available nodes. There are two kinds of nodes: *closed* and *open*. A node is called closed if there not exist new paths emanating from it; otherwise it is called open. The process ends when all the nodes are closed.

3. Lower and Upper Bounds

Let $R_{pq} = \{ i \text{ such that } l_i \leq p \text{ and } w_i \leq q \}$, that is, the set of the rectangular pieces that can be cut from an intermediary rectangle (p,q).

For any rectangle (p,q), the *homogeneous cutting patterns* (constituted of only one type of rectangular piece), can trivially be obtained. Therefore a lower bound is quickly determined by:

$$\pounds(p,q) = max \{ \pi_i . [p/w_i] . [q/l_i], \; i \in R_{pq} \}.$$

The lower bound for any rectangle (p,q) is always updated when the succeeding rectangles provide better homogeneous patterns, that is, ever that $\pounds(r,q) + \pounds(p-r,q) > \pounds(p,q)$ then $\pounds(p,q)$ is replaced by $\pounds(r,q) + \pounds(p-r,q)$. Hence, the lower bound for the ancestor nodes should also be updated (it can happen that other pattern provides a better lower bound for the ancestor nodes).

An upper bound can also be easily determined by considering the relaxed problem for the rectangle (p,q):

$$U(p,q) = \max \sum_{i \in R_{pq}} \pi_i a_i$$

$$\text{subject to: } \sum_{i \in R_{pq}} (w_i l_i) a_i \leq (p.q)$$

$$a_i \geq 0, \; i \in R_{pq}$$

All the feasible cutting patterns on (p,q) have to satisfy the above constraint, which imposes that the area cut by the pieces is less than or equal to the area of the whole rectangle (p,q). Note that if the integer condition is imposed on a_i, the above problem still provides an upper bound for the objective function and it becomes a knapsack problem. However, we need an upper bound easy to be computed, because it may be possible that thousands of nodes need to be examined. Thus the integer condition is not imposed and the above problem is trivially solved by:

$$U(p,q) = (p.q). \max \{ \pi_i/(w_i l_i), \; i \in R_{pq} \}.$$

These lower and upper bounds can be used to avoid unnecessary branching. For example, if $U(r,q) + U(p-r,q) \leq L(p,q)$ then a cut at r can be discarded without losing the optimality, since the best patterns for the rectangles (r,q) and $(p-r,q)$ can not be better than the already known pattern for (p,q) which gives $L(p,q)$. Furthermore, these bounds will be used to provide some heuristics.

4. Search Strategies

There are many ways to search a graph. Here we briefly comment on the main strategies. For more details see Pearl [1984].

The recursive procedure proposed by Herz [1972] is a depth

first search. To be precise, Herz used a version of depth-first search known as *backtracking*, which consists in cutting the newly open rectangle until all the generated rectangles be closed.

Another search strategy, generally more efficient than a depth-first search, is *best-first search* (*BF*). This strategy is used by the *AO* algorithm* (described in Pearl [1984] or Rich [1983]). Heuristic functions are used to evaluate paths and nodes to elect the best node to be first expanded. The best-first strategy considers all the earlier open nodes to be expanded, in opposition to the depth-first strategy that considers only the newly open nodes.

A simple and popular heuristic strategy is *hill-climbing* (*HC*) that is based on local optimizations. This strategy is heuristic and is called irrevocable, because it elects the local best path and abandones the other suspended paths for ever and so, the optimal solutions could be not reached.

Pearl [1984] represented the three main search strategies; hill-climbing (*HC*), depth-first (*DF*) and best-first (*BF*) as three extreme points of a continuous spectrum of search strategies. The search strategies are characterized along the following two dimensions:

1. *Recovery of pursuit(R):* The degree to which a search strategy allows recovery of previously suspended alternatives.
2. *Scope of evaluation (S):* The number of alternatives considered in each decision.

Along the *R* dimension we find *HC* at one extreme, discarding all the alternative paths, and *DF* and *BF* at the other extreme, permitting to recover all the suspended paths. Along the *S* dimension we find *HC* and *DF* that consider only the recent alternatives, whereas *BF* examines before each decision all the available alternatives.

Depth-First and Hill-Climbing Combination

The DF/HC Algorithm

Here we present a hybrid heuristic search algorithm that combines depth-first and hill-climbing strategies.

The maximum level that the depth-first search is permitted is denoted by *DEPTH*.

Initial step:

Let *init* be the initial *open* node, representing the rectangular plate. Compute $\ell(init)$ and do *nod* = *init*.

Main step:

While *init* is open, do:

1. Generate the successors from *nod* until the *DEPTH* level, using a depth-first strategy. For each generated node, compute its lower and upper bounds and update the lower bounds on the ancestor nodes until *init* be reached. Case a node has not new paths emanating from it, then close it.

2. Choose the best path so far (we used the path which provides the higher value to $\ell(init)$). Discard all the remainder paths. Note that it is a hill-climbing strategy.

3. Select an *open* node in this path to update *nod*. (The higher upper bound can be used to update *nod*).

The *DF/HC heuristic solution* is that provides $\ell(init)$.

The optimal solution may not be reached by the *DF/HC* algorithm, since the hill-climbing strategy eliminates paths that, although locally are not so good, could be part of the optimal solution. With *DEPTH* = *infinite*, then we have the depth-first search and so the optimal solution can be found.

5. Heuristics

Herz [1972] showed that the cuts on a rectangle can be restricted to combinations of the pieces dimensions without losing the optimal solution. However, the possible vertical or horizontal dimensions combinations can be quite large, in such a way that the problem becomes computationally untreatable. Beasley [1985] proposed the following heuristic to limit the size of these combinations:

Let P be the set of the possible combinations of w_i: $i=1,\ldots,m$. This set will define the possible horizontal cuts (analogous to the possible vertical cuts), and let M be a limit on the number of elements of P. (M is determined by the machine capacity). The following procedure is used to redefine P:

H1:

1) Let $N = \{ 1,\ldots,m \}$. The set N indicates the pieces to be used to construct P.
2) $P = \{ x \;/\; x = \sum \alpha_i . w_i,\ 1 \le x \le W,\ \alpha_i \ge 0$ and integer $i \in N \}$
3) If $|P| \le M$, then stop with the current P; otherwise go to (4).
4) Define $w_j = min \{ w_i \;/\; i \in N \}$;
   $N = N - \{ j \}$
   and go to step (2).

Of course the optimal solution, in general, will not be reached by this heuristic. Indeed, Morabito [1989] presented some examples for which two or three-stage cutting patterns can produce better results, so this heuristic should be careful used.

Following we present other heuristics that are more based on the graph representation than the previous one.

Suppose that a new path is produced by a cut on a rectangle A generating two new rectangles B and C. After this cut, the best pattern for A is limited by: $U(B) + U(C)$. So, if $U(B) + U(C) \le E(A)$ this cut can be discarded without losing the optimal

solution. But if $U(B) + U(C)$ is only slightly larger than $L(A)$, it can be a signal that this cut is not so good. We define a heuristic in the search process that dicards all the cuts that a given percentage of $U(B) + U(C)$ is less than $L(A)$.

H2:

Discard the cuts such that:

$$\alpha \, (\, U(B) + U(C) \,) < L(A)$$

where α is a previously given percentage.

The lower bounds $L(B)$ and $L(C)$ provide a new feasible solution to the rectangle A, probably composed by two types of pieces. Note that the optimal solution can be lost now if we discard a cut such that $L(B) + L(C) \leq L(A)$. However, there is a hope that better solutions are obtained whenever two types of pieces are being used rather than only one type. So, we introduce a heuristic on the search process that discards the cuts which a given percentage of $L(B) + L(C)$ is less than $L(A)$.

H3:

Discard the cuts such that:

$$\beta \, (\, L(B) + L(C) \,) < L(A)$$

where β is previously chosen percentage.

6. Computational Results

The DF/HC algorithm togheter with the heuristics were programmed in PASCAL and run on a microcomputer (PC-XT, 8 MHz). To show the perfomance of the algorithm and the effects of the heuristics, we used two examples from the two-dimensional cutting literature (Herz[1972,pp.469] and Beasley[1985,pp.305]) which have been reproduced in table I. Furthermore, we randomly generated five examples which are presented in table 4. For every

example we used $\pi_i = w_i / l_i$.

In the result-tables, |P| and |Q| indicate the size of the sets of possible vertical and horizontal cuts respectively. Note that |P| + |Q| plus the number of homogeneous cuttings represent the branching degree of the initial node. Of course rules that avoid duplications of patterns and the heuristics reduce significantly those branchings. The heading time1 in tables represents the time to obtain the sets P and Q, while the time2 is time spent by the graph search.

The optimal solution published by Herz has the objective function equals to 12348, and the heuristic solution published by Beasley (M=150, |P|=125, |Q|=109) is 8868950. Tables 2 and 3 give results for these examples using the DF/HC algorithm. Note that the solution obtained by the *DF/HC* algorithm for the Herz's example is optimal for DEPTH=3, $0.95 \leq \alpha \leq 1.0$ and $\beta = 1.0$. For $\beta < 1.0$ it was impossible to obtain the optimal solution. Note also, that the *DF/HC* solution for the Beasley's example, using M=150, is better than the solution found by Beasley, who used dynamic programming with different discretizations.

HERZ (n=5)		BEASLEY (n=32)	
W	L	W	L
127	98	3000	3000
w	l	w	l
18	65	365	185
24	27	378	200
21	13	410	165
36	17	425	148
54	20	425	296
		439	116
		464	1006
		520	205
		520	350
		540	530
		549	1413
		549	1882
		553	496
		555	755
		555	496
		555	659
		567	473
		572	592
		572	975
		572	1175
		572	1575
		572	1390
		572	1490
		572	1590
		572	1690
		572	1890
		610	625
		660	490
		690	447
		949	445
		949	478
		970	463

TABLE 1 - EXAMPLES OF HERZ AND BEASLEY

DEPTH	|P|	|Q|	α	β	MC-DF SOLUTION	#NODES	TIME1 (sec)	TIME2 (sec)
	M = 100							
2	60	33	1.0	1.0	12132	625	1	10
3	60	33	1.0	1.0	12348	3791	1	48
3	60	33	0.99	1.0	12348	3149	1	45
3	60	33	0.95	1.0	12348	2417	1	38
3	60	33	0.90	1.0	11466	1	1	4
3	60	33	1.0	0.999	12132	9	1	5
3	60	33	1.0	0.99	12132	9	1	5
3	60	33	1.0	0.95	11466	1	1	4

TABLE 2 - SOLUTION OF HERZ'S EXAMPLE

DEPTH	|P|	|Q|	α	β	MC-DF SOLUTION	#NODES	TIME1 (sec)	TIME2 (sec)
	M = 100							
2	86	72	1.0	1.0	8918106	69	34	11
3	86	72	1.0	1.0	8918106	115	34	21
3	86	72	0.99	1.0	8918106	111	34	20
3	86	72	0.95	1.0	8806000	1	34	5
3	86	72	1.0	0.999	8918106	9	34	7
3	86	72	1.0	0.99	8806000	1	34	5
	M = 150							
2	86	115	1.0	1.0	8918106	69	39	14
3	86	115	1.0	1.0	8918106	115	39	25
3	86	115	0.99	1.0	8918106	111	39	25
3	86	115	0.95	1.0	8806000	1	39	5
3	86	115	1.0	0.999	8918106	9	39	8
3	86	115	1.0	0.99	8806000	1	39	5
	M = 200							
2	161	182	1.0	1.0	8922452	121	94	21
3	161	182	1.0	1.0	8944026	283	94	48
3	161	182	0.99	1.0	8944026	245	94	47
3	161	182	1.0	0.999	8938594	13	94	11
	M = 300							
3	285	280	1.0	1.0	8944026	851	108	131
3	285	280	0.99	0.999	8944026	13	108	14

TABLE 3 - SOLUTION OF BEASLEY'S EXAMPLE

Table 4 shows five uniform randomly generated problems, with

m=10 (ie, 10 types of demanded pieces) and the dimensions of a piece j belong to the intervals:

$$0.15 * W \leq w_j \leq 0.65 * W$$
$$0.15 * L \leq l_j \leq 0.65 * L$$

	1		2		3		4		5	
(W, L)	(100,156)		(253,294)		(318,473)		(501,556)		(750,806)	
(w_j, l_j) $j=1,10$	27	32	130	130	141	128	136	113	205	163
	16	50	49	134	109	151	881	177	121	257
	47	66	45	52	76	142	237	237	356	343
	37	46	64	134	204	124	184	163	276	236
	31	51	38	100	110	165	153	182	230	264
	47	51	121	146	142	168	233	183	350	266
	48	44	48	123	82	238	241	155	361	225
	49	45	83	79	169	75	246	160	369	232
	59	78	42	187	165	185	296	278	445	404
	21	87	112	92	162	184	103	310	155	449

TABLE 4 - RANDOM PROBLEMS

The computational results for these problems are presented in table 5. It was supposed that each piece value is equal to its area, ie, $\Pi_j = w_j . l_j$. The depth is limited by 3 (ie, DEPTH=3), and α=0.99 and β=0.999.

Note that only one of the solutions obtained by the *DF/HC* algorithm is not optimal.

PROB	|P|	|Q|	OPTIMAL SOLUTION	HC-DF SOLUTION	ONODES	TIME1 (sec)	TIME2 (sec)
	N = 200						
1	47	73	15024	15024	11	2	2
2	153	86	73176	72172	3	5	4
3	70	155	142817	142817	123	5	9
4	116	138	265768	265768	5	6	2
5	122	145	577882	577882	5	9	3

TABLE 6 - SOLUTION OF RANDOM PROBLEMS

7. Conclusions and Perspectives

The and-or-graph representation presented in this paper, for the two-dimensional cutting problem, permits a better view of the techniques in the literature and it opens possibilities to the use of other efficient graph searches which were not explored so far. For example, the AO^* algorithm or the *staged search*, described in Pearl [1984] or Rich [1983].

In this paper, we have also suggested a *hybrid* algorithm, that combines two different known search strategies: the depth first search (*DF*) and hill climbing (*HC*). Furthermore, heuristics were proposed to speed up the graph search.

Some computational results are presented to illustrate the performance of the algorithm. The *DF/HC* algorithm seems to be a good alternative whenever storage and computational effort are essential factors.

Therefore, the two-dimensional cutting problem that has usually been solved by operational research techniques, can also be benefit from the graph searches, commonly used in the artificial intelligence environment.

B. References

Beasley, J.E. [1985] - "Algorithms for Unconstrained Two Dimensional Guillotine Cutting", Journal of Operational Research Society 4, 297-306.

Christofides, N. and Whitlock, C. [1977] - "An Algorithm for Two-dimensional Cutting Problems", Operations Research 25, 30-44.

Gilmore, P.C. and Gomory, R.E. [1965] - "Multistage Cutting Stock Problems of Two and More Dimensions", Operations Research 13, 94-120.

Herz, J.C. [1972] - "Recursive Computational Procedure for Two-Dimensional Stock Cutting", IBM J. Res. Develop. 16, 462-469.

Morabito, R.N. [1989] - "Corte de Estoque Bidimensional", Dissertação de Mestrado, ICMSC-USP.

Morabito, R.N, Arenales, M.N. and Arcaro, V.F. [1989]- "An and-or-graph representation to generate cutting patterns for the two-dimensional cutting problem", Workshop on Combinatorial Optimization,Rio de Janeiro-Brazil.

Pearl, J. [1984] - *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley Publishing Company.

Rich, E. [1983] - *Artificial Intelligence*, International Student Edition - McGraw-Hill.