OR Spektrum manuscript No.
(will be inserted by the editor)

A computational study of LP-based heuristic algorithms for two-dimensional cutting stock problems

Ramon Alvarez-Valdes[1] , **Antonio Parajon**[2], **Jose M. Tamarit**[1]

[1] Department of Statistics and Operations Research. University of Valencia, Spain

[2] Department of Mathematics. National Autonomous University of Nicaragua

Abstract In this paper we develop and compare several heuristic methods for solving the general two-dimensional cutting stock problem. We follow the Gilmore-Gomory column generation scheme in which at each iteration a new cutting pattern is obtained as the solution of a subproblem on one stock sheet. For solving this subproblem, in addition to classical dynamic programming, we have developed three heuristic procedures of increasing complexity, based on GRASP and Tabu Search techniques, producing solutions differing in quality and in time requirements. In order to obtain integer solutions from the fractional solutions of the Gilmore-Gomory process, we compare three rounding procedures, rounding up, truncated branch and bound and the solution of a residual problem.

We have coded and tested all the combinations of algorithms and rounding procedures. The computational results obtained on a set of randomly generated test problems show their relative efficiency and allow the potential user to choose from among them, according to the available computing time.

Running title: LP-heuristics for cutting stock problems

Keywords: cutting stock, column generation, heuristics, GRASP, tabu search.

Corresponding author:

Ramon Alvarez-Valdes

Dept. Estadistica i Investigacio Operativa.

Universitat de Valencia

Doctor Moliner 50

46100 Burjassot SPAIN

e-mail: ramon.alvarez@uv.es

1 Introduction

A two-dimensional cutting stock problem can be defined as follows: A set of rectangular stock sheets of p different types is available. For each type of sheet S_p we know its length L_p and width W_p. From these sheets we have to cut smaller rectangular pieces of length l_i and width w_i, $i = 1, ..., m$ in order to satisfy a given demand for d_i pieces of each type. The objective is to minimize the total area of stock sheets required. This problem has a wide range of commercial and industrial applications. It appears in cutting wood to make furniture and cardboard to make boxes, as well as in the production of glass, plastic or metal pieces.

A sequence of cuts of a sheet into rectangular pieces is a cutting pattern. In this paper we impose two constraints on the cutting patterns that usually appear in real-life situations. First, the pieces have a fixed orientation. Second, only *guillotine* cuts, that is, cuts going from one edge of the rectangle to the opposite edge, are allowed. Another constraint appearing sometimes is the limitation in the number of stages in which the cuts are made. However, in this paper we do not limit this number of stages and allow *n-stage* patterns.

The two-dimensional cutting stock problem has been extensively studied. In 1965 Gilmore and Gomory [9] extended their previous work on one dimensional cutting problems [7,8] to the two-dimensional case, proposing a column generation scheme in which at each step a new column is produced by solving a generalized knapsack problem in which a new cutting pattern is generated. Much theoretical and applied work has followed the Gilmore-Gomory approach [3,16]. However, as Riehme et al. [17] point out, in general it is not suitable for instances with small order demands. Also, for large size problems the solution process can be extremely long. Therefore, Riehme et al. [17] propose an alternative approach for the two-stage version of the problem. Different approaches are used by Wang [18], who solves a linear problem on a reduced set of high quality cutting patterns, and Yanasse et al. [19], who develop an enumeration scheme.

In this paper we again take the Gilmore-Gomory approach and perform a computational study in which we use several algorithms to solve the column generation subproblem and several rounding procedures to obtain integer solutions. The objective of this work is to explore the possibility of combining these three elements into an algorithmic scheme to efficiently solve two-dimensional cutting stock problems of large size and all types of demands in moderate computing times.

In Section 2 we describe the Gilmore-Gomory column generation scheme. In Section 3 we use the dynamic programming algorithm proposed by Beasley [2] to exactly solve the column generation subproblem appearing at each iteration. In Section 4 we propose several new heuristic algorithms for the same subproblem. Section 5 contains the computational study and the discussion about the rounding procedures. Finally, in Section 6 we present the conclusions.

2 The Gilmore-Gomory column generation scheme

For the two-dimensional cutting stock problem considered here, Gilmore and Gomory [9] propose solving the following integer program:

$$Min \sum_{q \in Q} c_q x_q$$

$$s.t. \quad \sum_{q \in Q} a_{iq} x_q \geq d_i, \quad i = 1, .., m \tag{1}$$

$$x_q \geq 0 \; and \; integer, \forall \, q \in Q \tag{2}$$

where Q is the set of all feasible two-dimensional cutting patterns for all sheets S_p, x_q the number of times pattern q is used in the solution, a_{iq} the number of times piece i appears in pattern q, d_i the demand of piece i and c_q the cost of pattern q. Usually this cost corresponds to the cost of the sheet S_p, so we can denote it by c_p.

As the set of patterns Q cannot be completely described except for very small problems, they develop a column generation scheme that can be summarised as follows:

1. Generate an initial set $\widetilde{Q}$ of m cutting patterns, where each pattern contains one type of piece.
2. Solve the linear relaxation of the above formulated problem considering only the variables corresponding to patterns in $\widetilde{Q}$.
3. For each type of sheet S_p, solve the subproblem:

$$z_p = Max \sum_i \pi_i a_i$$
$$\text{s.t.} \quad \{a_1, ..., a_m\} \text{ is a feasible cutting pattern for } S_p$$

   where π is the vector of dual prices of the LP solution. If, for some p, $z_p > c_p$, then the column corresponding to that solution is added to $\widetilde{Q}$ and Step 2 is returned to in order to solve the enlarged LP problem. Otherwise, the current solution is rounded to get an integer solution and the process ends.

The question now is how to solve efficiently the subproblem of Step 3. We describe exact and approximate methods in the following sections. Note that, even if we solve the subproblem by exact methods, the whole procedure is heuristic, due to the rounding step. The quality of the integer solution will depend not only on the algorithm used to solve the subproblem, but also on the rounding procedure and on the structure of the demands.

3 Solving the column generation subproblem by dynamic programming

Gilmore and Gomory [10] presented a dynamic programming recursion for staged two-dimensional cutting as well as for general (non-staged) guillotine cutting. Herz [11] noted that there is an error in that paper relating to the general non-staged algorithm. Beasley [2] proposes a corrected procedure for this non-staged case.

Let L_0, W_0 be the dimensions of the stock rectangle being considered for cutting and l_i, w_i, v_i, the length, width and value of each type of piece i, $i = 1, ..., m$. Let

$$F(0, x, y) = max\{0, v_i \,|\, l_i \leq x, w_i \leq y, i = 1, ..., m\}.$$

Then, if $F(-,x,y)$ represents the optimal value for the general guillotine cutting of a rectangle of size (x,y), we have:

$$F(-,x,y) = max\{F(0,x,y);$$
$$F(-,x,y_1) + F(-,x,q(y-y_1)), y_1 \in W, y_1 \leq y/2;$$
$$F(-,x_1,y) + F(-,p(x-x_1),y), x_1 \in L, x_1 \leq x/2\},\ \ x \in L^0, y \in W^0.$$

where $L = \{x \,|\, x = \sum_{i=1}^{m} l_i a_i, 1 \leq x \leq L_0 - min\{l_i\}, a_i \geq 0 \text{ and integer}, i = 1,..,m\}$,

$W = \{y \,|\, y = \sum_{i=1}^{m} w_i b_i, 1 \leq y \leq W_0 - min\{w_i\}, b_i \geq 0 \text{ and integer}, i = 1,..,m\}$

$L^0 = L \bigcup \{L_0\}$

$W^0 = W \bigcup \{W_0\}$

$p(x) = max\{0, x_1 \,|\, x_1 \leq x, x_1 \in L\},\ x < L_0$

$q(y) = max\{0, y_1 \,|\, y_1 \leq y, y_1 \in W\},\ y < W_0$

As Beasley [2] points out, if $|L|$ or $|W|$ are large, the recursion becomes computationally difficult, and he develops a heuristic procedure redefining L and W to ensure that $|L|$ and $|W|$ are small.

4 Solving the column generation subproblem by heuristic methods

If dynamic programming cannot be used, the column generation subproblem may be solved approximately, obtaining at each step a feasible solution of cost $\tilde{z_p}$. That will speed up the process in two ways. First, the approximate algorithms are faster. Second, the total number of generated columns will be lower (the condition to include a new column is now $\tilde{z_p} > c_p$). Many heuristic algorithms have been developed for this problem [14,4,12,5,15,18]. In this section we propose three new heuristic algorithms. First, we describe a very fast constructive procedure. Then, we develop a fast GRASP algorithm and finally a more complex and time-consuming tabu search algorithm. These procedures will allow us to compare the quality of the solutions and the time requirements of simple and sophisticated solution methods. More details on these algorithms may be found in [1] but, for completeness, we will describe them in the following subsections.

4.1 A constructive algorithm

Step 0.- Initialization

Let $L = \{R\}$, the list of rectangles still to be cut. Initially, the original rectangle R.

Let $P = \emptyset$, the set of pieces already cut.

Let $v_T = 0$, the total value of pieces cut.

The set S of pieces to cut is ordered by non-increasing $r_i = \frac{v_i}{s_i}$ (value to surface).

Ties are broken by non-increasing v_i.

For each piece i, n_i is the number of pieces cut. Initially, $n_i = 0$.

Step 1.- While $L \neq \emptyset$,

Take the smallest rectangle of L, $R_k = (L_k, W_k)$ and set $L = L - \{R_k\}$

Step 2.- For each piece $i, i = 1, .., m$,

If $l_i \leq L_k, \ w_i \leq W_k, \ n_i \leq d_i$,

Consider the consequences of cutting the piece i at the bottom left hand corner of R_k:

Compute the estimates $e_1 = BK(R_1^i)$, where $R_1^i = (l_i, W_k - w_i)$,

and $e_2 = BK(R_2^i)$, where $R_2^i = (L_k - l_i, W_k)$,

obtained by a vertical guillotine cut.

Also compute $h_1 = BK(R_3^i)$, where $R_3^i = (L_k - l_i, w_i)$

and $h_2 = BK(R_4^i)$, where $R_4^i = (L_k, W_k - w_i)$,

obtained by a horizontal guillotine cut.

The total potential benefit of cutting piece i is: $B_i = v_i + max\{e_1 + e_2, h_1 + h_2\}$

Else, $B_i = 0$

Step 3.- For the piece j, such that $B_j = max_i\{B_i\}$

If $B_j > 0$:

Cut piece j at the bottom left hand corner of R_k

Set $P = P \bigcup \{j\}$, $n_j = n_j + 1$ and $v_T = v_T + v_j$

If $e_1 + e_2 \geq h_1 + h_2$, then

$$L = L \bigcup \{R_1^j\} \bigcup \{R_2^j\}$$

Else, $L = L \bigcup \{R_3^j\} \bigcup \{R_4^j\}$

If $B_j = 0$, rectangle R_k is waste.

Go to Step 1.

In Step 2, the estimates are obtained by solving for each rectangle $R = (L, W)$ the problem:

$$BK(R) = Max \sum_{i \in S^*} v_i x_i$$

$$s.t. \quad \sum_{i \in S^*} s_i x_i \leq LW$$

$$x_i \leq min\{b_i - n_i, \lfloor \frac{L}{l_i} \rfloor \lfloor \frac{W}{w_i} \rfloor\}, i = 1, .., m$$

where $S^* = \{i | l_i \leq L, w_i \leq W\}$. This knapsack problem on bounded variables is approximately solved by using a greedy algorithm proposed by Martello and Toth [13].

4.2 A GRASP algorithm

The GRASP (Greedy Randomized Adaptive Search Procedure) is an iterative procedure in which at each iteration a constructive and an improving phase are performed (see [6] for an introduction). In our case, the constructive phase follows the algorithm in the previous subsection, but in Step 3, instead of selecting the piece j with maximum potential benefit B_j, the piece is selected at random from $S_{max} = \{i | B_i \geq \delta B_j\}$, where δ is an acceptance parameter $(0 \leq \delta \leq 1)$.

The improvement phase considers the waste rectangles produced in the constructive process and merges them, if possible, with adjacent pieces in order to create new rectangles that could be cut, possibly with greater value. Two rectangles are adjacent if they share a side. The new rectangles produced in that way are cut following the constructive procedure described above, producing a new complete solution.

The GRASP algorithm repeats these construction and improving phases until it reaches a given number of iterations without improving the best known solution.

4.3 A Tabu Search algorithm

We have to define a move which preserves the guillotine cut structure. We take a pair of neighbouring rectangles R_i and R_j (sharing at least one vertex) and define the common region as the smallest rectangle containing both of them. We then determine the set of guillotine cuts nearest to that region and containing it. The new rectangle defined by these cuts is emptied and cut again, thereby producing a new complete solution.

At each iteration we select the move to make in a three-step procedure. First, a rectangle from the solution (piece or waste) is selected at random. Second, we consider all its neighbours, one at a time. Third, for each pair of rectangles we follow the process described above. Cutting the new rectangle is done several times by performing several GRASP iterations. The move to be made is selected as the alternative producing the maximum value solution.

For each move we keep in the tabu list the dimensions and position of the new rectangle that has been cut again and the type of piece cut at its bottom left hand corner. The list length changes dynamically in the interval $[0.5\sqrt{m}, 1.5\sqrt{m}]$ after a given number of iterations without improving the best known solution.

Tabu search moves perform GRASP iterations and each GRASP iteration is based on the constructive algorithm. Therefore, the core decision process is the evaluation of the potential benefit B_i of cutting a piece i from a rectangle R. We have designed a change in this evaluation that can be used for intensification and diversification purposes. In the search process, we keep the n best solutions obtained so far. If f_{ik} is the number of times piece i appears in solution k, and M is the total number of pieces in the n solutions, we define a modified value of piece i:

$$p_i = v_i(1 + \beta \sum_k \frac{f_{ik}}{M})$$

If parameter $\beta > 0$, the value of pieces appearing more often in the best solutions is increased and their presence is favored, in an intensification strategy. On the contrary, if $\beta < 0$, the pieces with higher frequencies will have lower values and other pieces will have more possibilities of appearing in the solutions, in a diversification strategy.

5 Computational results: the rounding-off problem

In order to compare the relative efficiency of the proposed algorithms for solving the column generation subproblems, we have used a set of randomly generated instances, following the generation scheme described in [17].

5.1 Test problems

The instances were generated using independent and uniformly distributed integers of various ranges. The number p of different stock sheets is chosen from [3,5]. The lengths L_p and widths W_p are taken from intervals [1400, 2000] and [700, 1000], respectively.

We have considered 3 classes of instances, according to the number of pieces. In class A the number of pieces, m, is chosen from [21,30], in class B from [31,40] and in class C from [41,50]. The lengths l_i and the widths w_i are taken from [140,1000] and [70,500], respectively. We have also considered 3 types of demands. Demands d_i of type S (small) are taken from the interval [1,25]. Demands of type L (large) are taken from [100,200]. Demands of type V (variable) are selected according to the result of a randomly chosen number z in [0,1]. If $z \leq 0.5$, d_i is chosen in [1,25]. If $z > 0.5$, d_i is chosen in

[100,200]. Combining the three classes of numbers of pieces and the three types of demands, we have 9 classes of test instances (AS, AL, AV, BS, BL, BV, CS, CL, CV) and we have generated 45 problems with 5 instances of each class.

5.2 Rounding-off procedures

The final LP solution of the Gilmore-Gomory procedure is usually fractional, but for most applications the number of times each pattern is used must be integer. As Farley [3] points out, this impasse can be overcome in many ways, ranging from rounding up the solution to solving the entire problem by using integer programming techniques. Riehme et al. [17] define a *residual problem* in which the continuous solution of the original problems is rounded down and the non-fulfilled demands have to be cut in a second phase by a different procedure. As their algorithm is designed for 2-stage cutting, it cannot be directly applied here, but their ideas have helped us to develop our own algorithm.

We have compared three strategies for this rounding procedure. First, the simple rounding-up of the solution. Second, solving the original problem by truncated branch-and-bound, using the columns obtained in the column generation process and stopping the search if it reaches 20000 LP iterations. Finally, we have developed a more complex procedure to solve the residual problem.

The RESIDUAL algorithm works as follows. Consider the stock sheets ordered by non-decreasing surface from S_1 to S_p. Let u_i be the unfulfilled demand for piece i, after rounding down the Gilmore-Gomory solution, and let D be the total surface of the unfulfilled demands.

a.- Constructive phase

While there is some $u_i > 0$,

Choose a stock sheet (the smallest S_i covering D, if any; S_p, otherwise)

Cut the sheet using the GRASP algorithm, with the number of pieces of each type i to be cut bounded by u_i.

Update the unfulfilled demands u_i and D.

b.- Improving phase

For each cutting pattern C obtained in Phase (a) on a sheet S_i:

Determine the smallest sheet S_j in which the pieces of C could fit according to their surface.

For each sheet k, from $k = j$ to $i - 1$, try to cut the pieces of C on S_k, using the GRASP algorithm.

If it is possible, change the cutting pattern C from S_i to S_k

Go to the next pattern.

5.3 Computational results

We have coded the dynamic programming algorithm described in Section 3 as well as the heuristic algorithms of Section 4. We have used them to solve the column generation subproblem in an accumulative way. First, we have used only the constructive algorithm of Section 4.1. Second, we used the constructive and GRASP algorithms. At each iteration we first try the constructive algorithm and, whenever it fails to produce a new column, we try the GRASP algorithm. Third, we used constructive, GRASP and Tabu Search algorithms at each iteration, calling on the more complex procedures only when it was necessary. Finally, we used constructive, GRASP and dynamic programming algorithms. The results of these four strategies appear in Tables 1 and 2. Table 1 gives the waste percentages of each method and Table 2 the CPU time in seconds on a Pentium II at 500 Mhz. In each case, we give the LP solution and the results of the three rounding procedures described in the previous subsection. The number of columns generated by each procedure appears in Table 3.

Some initial conclusions may be drawn from the information provided by these Tables. If we consider only the LP solutions provided by the four methods we observe first, in Table 4, that the quality of the linear solutions does not decrease, but actually increases with problem size. The reason may be that, for the same types of stock sheets, a larger number of pieces allows more combinations and results in an increase of the stock usage. Within each problem size (A, B, C), the larger waste percentages appear for instances with variable demands, the kind of problems addressed in [17]. A summary of waste

Table 1 Waste percentages for the four methods

Instance	Constructive				GRASP				Tabu Search				Beasley			
class	LP sol.	Round up	Trunc. B&B	Res. prob.	LP sol.	Round up	Trunc. B&B	Res. prob.	LP sol.	Round up	Trunc. B&B	Res. prob.	LP sol.	Round up	Trunc. B&B	Res. prob.
AS	4.42	27.87	12.83	7.02	2.96	25.02	10.76	6.16	2.67	24.34	10.25	6.32	2.32	25.11	8.44	6.67
AL	4.22	7.49	5.99	4.46	2.64	6.43	3.84	3.02	2.32	5.85	3.25	2.72	1.95	5.12	2.85	2.36
AV	4.98	10.00	6.64	5.40	3.66	9.06	5.04	4.25	3.52	8.44	5.01	4.07	3.23	7.89	4.69	3.91
BS	3.86	25.19	10.79	6.32	2.34	22.34	8.18	4.98	2.20	23.86	8.45	4.46	1.83	23.07	9.24	4.55
BL	4.02	6.53	5.08	4.23	2.44	5.52	3.33	2.79	2.27	5.37	3.36	2.49	1.85	4.84	3.03	2.14
BV	4.30	8.25	6.05	4.54	2.97	7.14	4.56	3.28	2.75	6.89	3.95	3.10	2.37	6.64	4.24	2.92
CS	3.27	23.83	11.08	4.86	2.17	22.61	12.07	3.77	1.91	22.38	10.87	4.17	1.58	22.89	11.40	3.57
CL	3.19	5.90	4.09	3.38	2.01	4.80	4.03	2.31	1.86	4.59	3.54	2.08	1.56	4.32	3.19	1.87
CV	3.85	8.00	5.65	4.11	2.54	7.12	4.85	2.99	2.30	6.70	5.21	2.68	2.00	6.61	4.64	2.47
Total	4.01	13.67	7.58	4.93	2.64	12.23	6.30	3.73	2.42	11.99	5.99	3.57	2.08	11.83	5.75	3.38

Table 2 Computing times for the four methods

Instance	Constructive				GRASP				Tabu Search				Beasley			
class	LP sol.	Round up	Trunc. B&B	Res. prob.	LP sol.	Round up	Trunc. B&B	Res. prob.	LP sol.	Round up	Trunc. B&B	Res. prob.	LP sol.	Round up	Trunc. B&B	Res. prob.
AS	22	22	84	64	250	250	323	276	1019	1019	1080	1041	3215	3215	3273	3241
AL	14	14	87	51	228	228	284	249	970	970	1027	994	2653	2653	2721	2681
AV	13	13	79	43	165	165	226	181	826	826	881	842	2123	2123	2182	2138
BS	52	52	104	147	662	662	718	716	1721	1721	1819	1771	5627	5627	5905	5679
BL	53	53	114	169	1004	1004	1061	1061	1941	1941	2047	2002	10007	10007	10173	10077
BV	77	77	145	195	870	870	959	914	2787	2787	2921	2831	8183	8183	8405	8235
CS	172	172	233	442	1207	1207	1547	1321	3782	3782	4081	3889	13399	13399	13920	13510
CL	175	175	245	456	1344	1344	1615	1455	3285	3285	3576	3404	12599	12599	12967	12721
CV	144	144	207	393	1585	1585	1841	1682	4278	4278	4580	4386	12456	12456	12923	12547
Total	80	80	144	218	813	813	953	873	2290	2290	2446	2351	7807	7807	8052	7870

Table 3 Columns generated by each method

Instance	Constructive	GRASP			Tabu Search				Beasley			
class	Total	Constr.	GRASP	Total	Constr.	GRASP	Tabu.	Total	Constr.	GRASP	Dyn. Pr.	Total
AS	107	156	30	185	159	50	5	214	160	61	47	269
AL	86	127	32	159	132	55	5	192	133	63	45	240
AV	91	136	28	163	138	40	5	183	138	51	34	224
BS	158	253	49	303	252	68	4	324	258	104	63	425
BL	157	240	67	308	241	80	4	325	250	108	85	444
BV	199	284	52	336	276	74	5	356	269	106	74	448
CS	250	353	68	421	363	117	6	485	364	164	105	633
CL	269	370	86	456	374	114	4	492	376	157	101	634
CV	246	374	75	449	376	110	6	491	377	158	84	619
Total	174	255	54	309	257	79	5	340	258	108	71	437

percentages and running times appears in Table 5, in which we can observe that small increases in quality are obtained at a very high computational cost.

Table 4 Waste percentages of LP solutions for different problem sizes

Instance size	Constructive	GRASP	Tabu Search	Beasley
A	4.54	3.09	2.84	2.50
B	4.06	2.58	2.41	2.02
C	3.44	2.24	2.02	1.89

Table 5 Waste percentages and running times of LP solutions for the four methods

Method	Waste percentage	Running time
Constructive	4.01	80
GRASP	2.64	813
Tabu Search	2.42	2290
Beasley	2.08	7807

Other interesting conclusions can be obtained by looking at the results of rounding procedures, summarized in Table 6. The choice of an adequate rounding method is absolutely critical, because the quality of the LP solution, obtained in a costly process, can be completely lost at the rounding phase. The performance of the three procedures studied is very different. Rounding up the fractional solutions produces on average an increase of more than a 9% in the waste percentages. As expected, the results are specially bad for small demand problems, with an average increase of over 20% , but even for large demand problems the average increase exceeds 3%. The second rounding procedure, truncated branch and bound on the generated columns, improves those results, but it is clearly beaten by the procedure consisting of rounding down the fractional solution and then solving the residual problem. It seems that allowing the presence in the solution of new cutting patterns, produced by a residual problem, is more efficient than forcing integer values for the columns generated in the Gilmore-Gomory process.

Table 6 Waste percentages of LP solutions and rounding procedures

Method	LP solution	Rounding up	Truncated branch & bound	Residual problem
Constructive	4.01	13.67	7.58	4.93
GRASP	2.64	12.23	6.30	3.73
Tabu Search	2.42	11.99	5.99	3.57
Beasley	2.08	11.83	5.75	3.38

In order to choose a complete procedure to get integer cutting solutions we can look at Table 7, in which we compare the waste percentages and running times of the four methods, using the third rounding procedure. Two methods appear to be the best choices, depending on the available computing time. If a fast solution is needed, the constructive method provides it with a waste percentage that is, on average, 1.55% worse than the best solution obtained by dynamic programming, requiring less than 4' of CPU. If some more computing time is available, the method based on the GRASP algorithm requires, on average, less than 15' and gets good quality solutions, less than 0.4% away from the best. Tabu Search and dynamic programming algorithms are too time-consuming.

Table 7 Waste percentages and running times of integer solutions for the four methods

Method	Waste percentage	Running time
Constructive	4.93	218
GRASP	3.73	873
Tabu Search	3.57	2351
Beasley	3.38	7870

6 Conclusions

This paper explores the use of the Gilmore-Gomory column generation scheme for solving medium-large size two-dimensional cutting problems. We propose some efficient heuristic algorithms for the column generation subproblem and a rounding procedure based on the solution of a residual problem. The computational results show that good quality integer solutions can be obtained in moderate computing times.

Another line of research would be that of looking for optimal solutions to this problem. The application of some of the ideas developed in this paper to this new objective is currently being investigated.

References

1. Alvarez-Valdes R, Parajon A, Tamarit JM (2000) A tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems. Computers and Operations Research (in press).
2. Beasley JE (1985) Algorithms for Unconstrained Two-Dimensional Guillotine Cutting. Journal of the Operational Research Society 36:297-306
3. Farley AA (1988) Practical adaptations of the Gilmore-Gomory approach to cutting stock problems. OR Spektrum 10:113-123
4. Fayard D, Zissimopoulos V (1995) An approximation algorithm for solving unconstrained two-dimensional knapsack problems. European Journal of Operational Research 84:618-632
5. Fayard D, Hifi M, Zissimopoulos V (1998) An efficient approach for large-scale two-dimensional guillotine cutting stock problems. Journal of the Operational Research Society 49:1270-1277
6. Feo T, Resende MGC (1995) Greedy Randomized Adaptive Search Procedures. Journal of the Global Optimization 2:1-27
7. Gilmore PC, Gomory RE (1961) A linear programming approach to the cutting-stock problem. Operations Research 9:849-859
8. Gilmore PC, Gomory RE (1963) A linear programming approach to the cutting-stock problem-Part II. Operations Research 11:863-888

9. Gilmore PC, Gomory RE (1965) Multistage cutting stock problems of two and more dimensions. Operations Research 13:94-120

10. Gilmore PC, Gomory RE (1966) The theory and computation of knapsack functions. Operations Research 14:1045-1074

11. Herz JC (1972) A recursive computing procedure for two-dimensional stock cutting. I.B.M. Journal of Research Development 16:462-469

12. Hifi M (1997) The DH/KD algorithm: a hybrid approach for unconstrained two-dimensional cutting problems. European Journal of Operational Research 97:41-52

13. Martello S, Toth P (1990) Knapsack problems. Algorithms and Computer Implementations. John Wiley & Sons Ltd. Chichester

14. Morabito RN, Arenales MN, Arcaro VF (1992) An and-or-graph approach for two-dimensional cutting problems. European Journal of Operational Research

15. Morabito RN, Arenales MN (1996) Staged and constrained two-dimensional guillotine cutting problems: an AND/OR-graph approach. European Journal of Operational Research 94:548-560

16. Morabito RN, Garcia V (1998) The cutting stock problem in a hardboard industry: a case study. Computers and Operations Research 25:469-485

17. Riehme J, Scheithauer G, Terno J (1996) the solution of two-stage guillotine cutting stock problems having extremely varying order demands. European Journal of Operational Research 91:543-552

18. Wang PY (1983) Two algorithms for constrained two-dimensional cutting stock problems. Operations Research 31:573-586

19. Yanasse HH, Zinober ASI, Harris RG (1991) Two-dimensional cutting stock with multiple stock sizes. Journal of Operational Research Society 42:673-683