J. Opl Res. Soc. Vol. 36, No. 4, pp. 297–306, 1985
Printed in Great Britain.

0160-5682/85 $3.00 + 0.00

Algorithms for Unconstrained Two-Dimensional Guillotine Cutting

J. E. BEASLEY
Department of Management Science, Imperial College, London

In this paper we consider the unconstrained, two-dimensional, guillotine cutting problem. This is the problem that occurs in the cutting of a number of rectangular pieces from a single large rectangle, so as to maximize the value of the pieces cut, where any cuts that are made are restricted to be guillotine cuts.

We consider both the staged version of the problem (where the cutting is performed in a number of distinct stages) and the general (non-staged) version of the problem.

A number of algorithms, both heuristic and optimal, based upon dynamic programming are presented. Computational results are given for large problems.

Key words: cutting, dynamic programming, heuristic

INTRODUCTION

The unconstrained, two-dimensional cutting problem is the problem of cutting from a single plane rectangular piece a number of smaller rectangular pieces, of a given size and each with a given value, so as to maximize the value of the pieces cut (there being no constraint on the number of pieces of each size that result from the cutting).

The problem appears in the cutting of steel plates into required sizes, in the cutting of wood sheets to make furniture and in the cutting of cardboard into boxes. The related problem of minimizing the amount of waste produced by the cutting can be converted into this problem by making the value of all pieces equal to their areas.

Whilst the general two-dimensional cutting problem given above has been considered by relatively few authors in the literature[1] restricted versions of the problem have been considered by a number of authors. There are two types of restrictions that are used:

(a) The first type of restriction is to consider only guillotine cuts—a guillotine cut on a rectangle being a cut from one edge of the rectangle to the opposite edge which is parallel to the two remaining edges. Guillotine cutting problems have been considered by Gilmore and Gomory[2,3] who presented a dynamic-programming procedure, and by Herz[4] and Christofides and Whitlock,[5] who presented tree-search procedures. Heuristic algorithms for the problem have also been developed.[6,7]

(b) The second type of restriction is to limit the cutting that occurs to a number of 'stages'. Regarding any two adjacent edges of the rectangle to be cut as x and y-axes, as shown in Figure 1, then the cuts at the first stage are restricted to be guillotine cuts parallel to one axis (e.g. the x-axis). The cuts at the second stage are restricted to be guillotine cuts parallel to the other axis (y-axis); the cuts at the third stage are restricted to be guillotine cuts parallel to the original axis (x-axis); etc. Figure 1 illustrates a three-stage cutting pattern where the cut direction at the first stage is parallel to the x-axis.

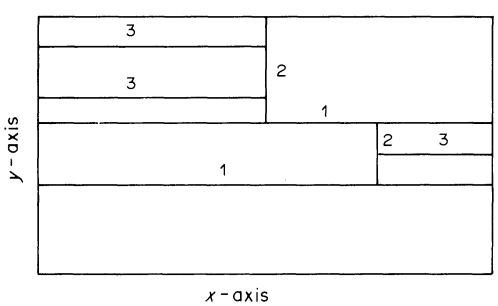

FIG. 1. *Three-stage cutting. Numbers by the cuts are the stage at which the cut is made.*

Staged cutting problems have been considered by Gilmore and Gomory,[2,3] Hahn[8] and Farley.[9,10] Gilmore and Gomory[2] were primarily concerned with the two-stage problem, which they formulated in two ways: as a generalized knapsack problem soluble as a set of one-dimensional knapsack problems and as a staged linear-programming problem. Hahn[8] considered three-stage problems, where any cuts at the third stage produced pieces of identical dimensions and there were defects in the rectangle being cut. She used an extension of the approach for two-stage cutting that Gilmore and Gomory[2] developed. Farley[9,10] considered the problem of ensuring that any cuts that are made are greater than a critical distance apart and presented modifications to the Gilmore and Gomory[2] algorithm.

Note here that virtually all the approaches in the literature for two-dimensional guillotine cutting involve the use of dynamic-programming.[11–13]

In this paper we present a number of algorithms for unconstrained two-dimensional guillotine cutting. We start by considering staged cutting.

STAGED CUTTING

Gilmore and Gomory[3] presented a dynamic-programming recursion for staged two-dimensional cutting. Herz[4] has noted that there is an error in that paper relating to an algorithm for general (non-staged) guillotine cutting. In fact, we show below that there is a further error in that paper relating to the dynamic-programming recursion given for staged two-dimensional cutting. In this section we develop a correct dynamic-programming recursion for the staged two-dimensional cutting problem.

Let the large rectangle A_0 which is to be cut be of length L_0 and width W_0, and let m smaller rectangular pieces which can be cut from it be given, where piece i is of length L_i, width W_i and value v_i (≥ 0). Note that any piece cut from A_0 which is not of size (L_i, W_i), for some i, is considered to be of value zero. We assume the following.

(1) L_0, W_0 and L_i, W_i $i = 1, \ldots, m$ are integers.

(2) The orientation of the pieces is considered to be fixed (i.e. a piece of length s and width t is not the same as a piece of length t and width s ($s \neq t$)). This is not a serious restriction, since if a piece (s, t) can be cut with either orientation, we treat it as two different pieces [(s, t) and (t, s)], each of which has a fixed orientation.

(3) Any cuts that are made are infinitely thin. Again, this is not a serious restriction since it is well known that a problem in which all cuts are width d (integer) can be converted into a problem in which the cuts are infinitely thin by adding d to L_0, W_0 and L_i, W_i $i = 1, \ldots, m$.

The problem then is to develop a staged cutting pattern for A_0 such that no more than n stages are used and we maximize the value of the pieces cut.

Note here that there are no constraints upon the number of pieces of each type that are cut from A_0. Christofides and Whitlock[5] have presented a tree-search procedure for the optimal solution of two-dimensional guillotine cutting problems involving constraints of this type. Their approach can be adapted to staged cutting with the aid of the dynamic-programming recursion for staged cutting developed below.

In staged cutting, the cuts alternate at each stage between being parallel to the x-axis and being parallel to the y-axis. Hence we can associate with each stage a cut direction. Note, however, that we do not require a cut to be made at each stage (i.e. at a certain stage we may not make a cut in the appropriate (stage-dependent) cut direction).

To formulate the problem, let $L = [1,2, \ldots, L_0 - 1]$ be the set of possible lengths for any cuts parallel to the y-axis and let $W = [1, 2, \ldots, W_0 - 1]$ be the set of possible widths for any cuts parallel to the x-axis. Define:

$F(k, x, y)$ = the value of an optimal k-stage cut of a rectangle of size (x, y) where the first-stage cut direction is parallel to the x-axis;

$G(k, x, y)$ = the value of an optimal k-stage cut of a rectangle of size (x, y) where the first-stage cut direction is parallel to the y-axis.

A value for k of zero in the above definition corresponds to a trimming of at most one piece from the rectangle (x, y). We can identify four trimming situations:

(1) no trimming allowed:

$$F(0, x, y) = \max(0, v_i \mid L_i = x, \quad W_i = y \quad i = 1, \ldots, m), \tag{1}$$

(2) allow trimming parallel to the x-axis but no trimming parallel to the y-axis:

$$F(0, x, y) = \max(0, \mathrm{v_i} \mid L_i = x, \quad W_i \leqslant y \quad i = 1, \ldots, m), \tag{2}$$

(3) allow trimming parallel to the y-axis but no trimming parallel to the x-axis:

$$F(0, x, y) = \max(0, v_i \mid L_i \leqslant x, W_i = y \quad i = 1, \ldots, m), \tag{3}$$

(4) allow trimming parallel to both the x-axis and the y-axis:

$$F(0, x, y) = \max(0, v_i \mid L_i \leqslant x, W_i \leqslant y \quad i = 1, \ldots, m). \tag{4}$$

We use T to distinguish the four trimming cases given above (e.g. $T = 3$ corresponds to the third trimming case). Note here that the values $F(0, x, y)$ and $G(0, x, y)$ are identical.

We can now develop a dynamic-programming recursion for $F(n, L_0, W_0)$ and $G(n, L_0, W_0)$ as follows. Consider the k-stage cutting of a rectangle (x, y) where the first-stage cut direction is parallel to the x-axis; then there are only three alternatives for the optimal cutting pattern:

(a) No staged cuts are made but a piece is trimmed (if possible) from (x, y)—as in Figure 2(a).

(b) There is at least one first-stage cut parallel to the x-axis (at some $\mathrm{Y_1} \in W$)—as in Figure 2(b). If we are allowing trimming parallel to the x-axis ($T = 2$ or $T = 4$), then (by symmetry) we need only consider cuts at values of y_1 satisfying $y_1 \leqslant y/2$.

(c) There are no first-stage cuts parallel to the x-axis but at least one second-stage cut parallel to the y-axis (at some $x_1 \in L$)—as in Figure 2(c). In this case we have a cutting pattern where the first-stage cut direction is parallel to the y-axis and there are $k - 1$ stages to the cutting pattern. Hence

$$F(k, x, y) = \max[F(0, x, y); F(k, x, y_1) + F(k, x, y - y_1), y_1 \in W, y_1 \leqslant K_1(y); G(k - 1, x, y)], \tag{5}$$

where

$$K_1(y) = y/2 \quad \text{if} \quad T = 2 \text{ or } T = 4$$
$$= y - 1 \text{ otherwise.}$$

A similar argument to the one given above can be used to show that

$$G(k, x, y) = \max[G(0, x, y); G(k, x_1, y) + G(k, x - x_1, y), x_1 \in L, x_1 \leqslant K_2(x); F(k - 1, x, y)], \tag{6}$$

where

$$K_2(x) = x/2 \quad \text{if} \quad T = 3 \quad \text{or} \quad T = 4$$
$$= x - 1 \text{ otherwise.}$$

Equations (5) and (6) apply for any $k \geqslant 1$ and any rectangle (x, y).

These equations are the basic dynamic-programming recursion for the optimal k-stage cutting of a rectangle and can be used to calculate the value of an optimal n-stage cutting pattern for A_0 [$F(n, L_0, W_0)$ or $G(n, L_0, W_0)$, depending upon the first-stage cut direction specified, or $\max[F(n, L_0, W_0), G(n, L_0, W_0)]$ if the first-stage cut direction is unspecified]. Equations (1)–(4) provide initial conditions for the recursion.

Note here that discovering the nature of the cutting pattern that gives the optimal value is a simple matter if pointers are used which indicate, for each (k, x, y), the terms in the recursion that led to the values $F(k, x, y)$ and $G(k, x, y)$. A simple backtracking routine can then be used to unravel the cutting pattern associated with the optimal value.

We said before that the dynamic-programming recursion for k-stage cutting given by Gilmore and Gomory[3] [p. 1073, equation (35)] was incorrect. This becomes apparent when one compares the recursion developed above with the one given in their paper. The error arises because they fail

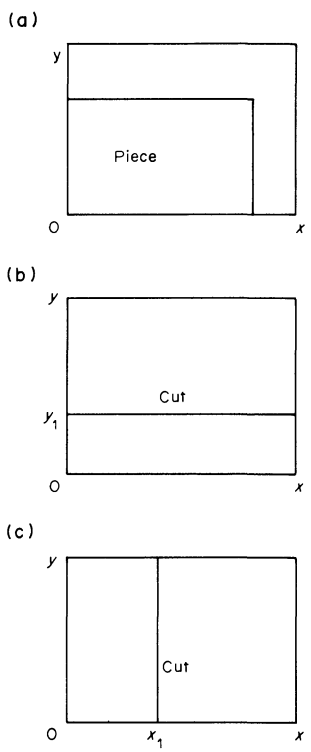

FIG. 2. (*a*) *Trim one piece.* (*b*) *First-stage cut parallel to the x-axis.* (*c*) *No first-stage cut, but a second-stage cut parallel to the y-axis.*

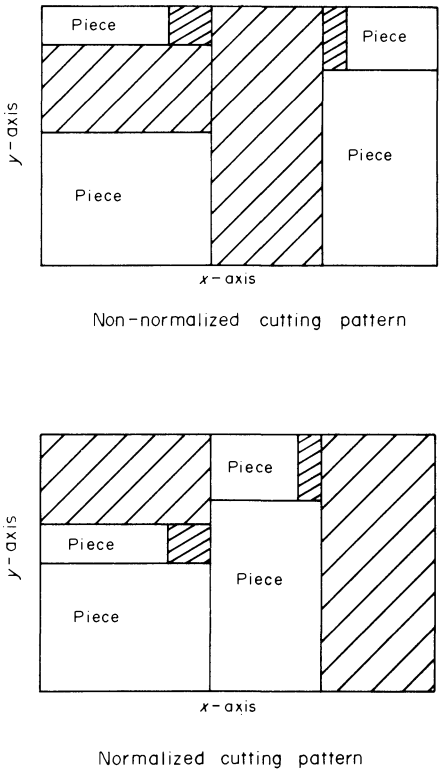

FIG. 3. *Normalized cutting pattern.*

to develop different functions for different first-stage cut directions. Whilst in k-stage cutting the first-stage cut direction is fixed (once specified), the recursion given in Gilmore and Gomory alters the first-stage cut direction with k (e.g. in that recursion with $k = 3$, the first-stage cut direction is parallel to the y-axis, but with $k = 4$, the first-stage cut direction is parallel to the x-axis).

In the next section we show how the dynamic-programming recursion for staged cutting given above can be enhanced computationally through the use of normal patterns.

NORMAL PATTERNS

Normal patterns were used by Herz[4] (who called them canonical dissections) and Christofides and Whitlock[5] and are based on the fact that, given a cutting pattern, any piece (or cut) in that pattern can be moved as shown in Figure 3 until both the left-hand edge and the bottom edge of all pieces are adjacent to a cut (or A_0). Note here that normalization preserves any stage property in the original cutting pattern.

Note also that a cut at a value of $x > L_0 - \min(L_i \mid i = 1, \ldots, m)$ can have no piece lying to the right of the cut and so can be regarded as being associated with the trimming of some piece lying to the left of the cut.

Hence the set L of possible lengths for any cuts parallel to the y-axis can be changed from $L = [1, 2, \ldots, L_0 - 1]$ to the set

$$L = \left[x \mid x = \sum_{i=1}^{m} L_i a_i,\ 1 \leqslant x \leqslant L_0 - K_3,\ a_i \geqslant 0 \quad \text{and} \quad \text{integer } i = 1, \ldots, m \right], \tag{7}$$

where

$$K_3 = \min(L_i \mid i = 1, \ldots, m) \quad \text{if} \quad T = 3 \quad \text{or} \quad T = 4$$
$$= 1 \text{ otherwise.}$$

Equation (7) essentially says that if $x \in L$, there exists a set of pieces whose lengths add up to x. The equivalent definition for W is

$$W = \left[y \mid y = \sum_{i=1}^{m} W_i b_i,\ 1 \leqslant y \leqslant W_0 - K_4,\ b_i \geqslant 0 \quad \text{and} \quad \text{integer } i = 1, \ldots, m \right], \tag{8}$$

where

$$K_4 = \min(W_i \mid i = 1, \ldots, m) \quad \text{if} \quad T = 2 \quad \text{or} \quad T = 4$$
$$= 1 \text{ otherwise.}$$

Note that L and W are defined with respect to the pieces that can be cut from A_0 and are easily calculated.[5] Normal patterns can be used to enhance the basic dynamic-programming recursion given previously in the following way.

Let

$$p(x) = \max(0, x_1 \mid x_1 \leqslant x,\ x_1 \in L) \quad x < L_0 \tag{9}$$

$$q(y) = \max(0, y_1 \mid y_1 \leqslant y,\ y_1 \in W) \quad y < W_0, \tag{10}$$

i.e. $p(x)$ is the length nearest to x in the normalized set of lengths L [$p(x) \leqslant x$], where we define $p(x) = 0$ if no such length exists and $q(y)$ is the width nearest to y in the normalized set of widths W[$q(y) \leqslant y$], where we define $q(y) = 0$ if no such width exists. To simplify the algorithm, we define $p(L_0) = L_0$ and $q(W_0) = W_0$. Note that this may not be strictly necessary since, for example, there may not exist a set of piece lengths which add to L_0 (and similarly for W_0). Then we claim that:

$$F[k, p(x), q(y)] \geqslant F(k, x, y) \quad k \geqslant 0;\ x = 1, 2, \ldots, L_0;\ y = 1, 2, \ldots, W_0 \tag{11}$$

(where we introduce the convention that any term with a zero value for x or y in $F(k, x, y)$ is taken to be zero).

Equation (11) is true because the optimal k-stage cutting pattern for (x, y) can be normalized into a k-stage cutting pattern for $[p(x), q(y)]$, as noted above, so that $F[k, p(x), q(y)] \geqslant F(k, x, y)$ (i.e. as we have a feasible solution for $[p(x), q(y)]$ of value $F(k, x, y)$, the optimal solution $F[k, p(x), q(y)]$ must either equal or exceed this). A similar inequality to equation (11) applies for $G(k, x, y)$, namely:

$$G[k, p(x), q(y)] \geqslant G(k, x, y) \quad k \geqslant 0;\ x = 1, 2, \ldots, L_0;\ y = 1, 2, \ldots, W_0 \tag{12}$$

(where we have the convention that any term with a zero value for x or y in $G(k, x, y)$ is taken to be zero).

The recursion for $F(k, x, y)$ [equation (5)] can now be modified to be

$$F(k, x, y) = \max[F(0, x, y);\ F(k, x, y_1) + F[k, x, q(y - y_1)],\quad y_1 \in W,\quad y_1 \leqslant K_1(y);\ G(k - 1, x, y)]. \tag{13}$$

Here we have modified the second term in the recursion. That term arose from considering the optimal k-stage cutting pattern of (x, y) to consist of some first-stage cut parallel to the x-axis. We now have that first-stage cut at some $y_1 \in W$ (the normalized set of widths), and the two pieces left after this cut of (x, y) of sizes (x, y_1) and $(x, y - y_1)$ are considered for optimal k-stage cutting. The optimal k-stage cut of $(x, y - y_1)$ consists of an optimal k-stage cut of $[x, q(y - y_1)]$, with a first-stage cut at $[y_1 + q(y - y_1)]$ separating the normal cutting pattern of size $[x, q(y - y_1)]$ from an area of size $[x, y - y_1 - q(y - y_1)]$ that is left with no piece cut out from it—see Figure 4.

In a similar fashion, we can modify the recursion for $G(k, x, y)$ [equation (6)] to

$$G(k, x, y) = \max[G(0, x, y);\ G(k, x_1, y) + G[k, p(x - x_1), y],\ x_1 \in L,\ x_1 \leqslant K_2(x);\ F(k - 1, x, y)]. \tag{14}$$

It is now clear that, in calculating $F(n, L_0, W_0)$ and $G(n, L_0, W_0)$ using equations (13) and (14), it is not necessary to calculate $F(k, x, y)$ and $G(k, x, y)$ for all values of x and y and all $k (\leqslant n)$ but that it is sufficient to restrict attention to $x \in L^0$ (where $L^0 = L \cup [L_0]$) and $y \in W^0$ (where $W^0 = W \cup [W_0]$) and all $k (\leqslant n)$. This is so since any cuts made in the optimal n-stage cutting pattern

for (L_0, W_0) can be normalized to occur at $x \in L$ and $y \in W$, and the optimal cutting of any rectangle (x, y) is dominated by the optimal cutting of the rectangle $[p(x), q(y)]$ [equations (11) and (12)]. Hence we have the final modified dynamic-programming recursion

$$F(k, x, y) = \max[F(0, x, y); F(k, x, y_1) + F[k, x, q(y - y_1)], y_1 \in W, y_1 \leqslant K_1(y); G(k - 1, x, y)] \quad k \geqslant 1, x \in L^0, y \in W^0. \quad (15)$$

$$G(k, x, y) = \max[G(0, x, y); G(k, x_1, y) + G[k, p(x - x_1), y], x_1 \in L, x_1 \leqslant K_2(x); F(k - 1, x, y)] \quad k \geqslant 1, x \in L^0, y \in W^0. \quad (16)$$

It is clear that the modified recursion given above is computationally more effective than the basic recursion given previously [equations (5) and (6)]. Indeed, the calculation of an optimal n-stage cutting pattern using the modified recursion involves only $0[n \mid L \parallel W \mid (\mid L \mid + \mid W \mid)]$ operations and requires a storage capacity for $0(\mid \mathrm{L} \parallel \mathrm{W} \mid)$ numbers.

However, it is also clear that the above recursion will become computationally infeasible if $\mid \mathrm{L} \mid$ or $\mid \mathrm{W} \mid$ is large. Typically this will happen if at least one piece (i, say) is small compared with A_0 (L_i small compared with L_0 or W_i small compared with W_0) and $A_0 = (L_0, W_0)$ is also large. Many practical problems are of this type.

In such a case it is possible to modify the optimal algorithm given above into a computationally feasible heuristic algorithm. For simplicity we shall deal with this modification in the context of general (non-staged) guillotine cutting, which we consider in the next section, but note here that the approach followed there can also be applied to the staged cutting algorithm presented above.

GENERAL GUILLOTINE CUTTING

Dynamic-programming recursion

Herz[4] presented a recursive (tree-search) procedure for the problem of general (non-staged) guillotine cutting and compared his algorithm with the algorithm of Gilmore and Gomory.[3] His procedure is heavily dependent upon the availability of an upper bound $u(x, y)$ on the value of the optimal cutting pattern for all rectangles (x, y) $x = 1, 2, \ldots, L_0$, $y = 1, 2, \ldots, W_0$. Whilst this upper bound is easily available for problems involving the maximization of area cut from A_0 [when $u(x, y) = xy$], such as the problem solved by Hertz, it is clear that obtaining an upper bound $u(x, y)$ for problems with a different objective is a non-trivial task.

Accordingly we would expect that a reasonable approach to obtaining the optimal solution to the general (non-staged) guillotine cutting problem would be a dynamic-programming recursion of the kind presented by Gilmore and Gomory[3] (also Gilmore and Gomory,[2] p. 100). Because very few computational results have been reported in the literature for general guillotine cutting, we decided to implement and report computationally on such a recursion. As the general guillotine cutting problem can be viewed as a staged cutting problem, with the number of stages unknown, we can adapt the recursion given before for staged cutting.

Let $F(0, x, y)$ be as defined by equation (4) (i.e. $T = 4$); then, letting $F(-, x, y)$ represent the optimal value for the general guillotine cutting of a rectangle of size (x, y), we have that

$$F(-, x, y) = \max[F(0, x, y); F(-, x, y_1) + F[-, x, q(y - y_1)], y_1 \in W, y_1 \leqslant y/2; F(-, x_1, y) + F[-, p(x - x_1), y], x_1 \in L, x_1 \leqslant x/2] \quad x \in L^0, y \in W^0. \quad (17)$$

See also Gilmore and Gomory[2] (p. 100) and Herz.[4]

As for staged cutting, if $\mid L \mid$ or $\mid W \mid$ is large, then the recursion [equation (17)] becomes computationally infeasible since it involves $0[\mid L \parallel W \mid (\mid L \mid + \mid W \mid)]$ operations and requires a storage capacity for $0(\mid \mathrm{L} \parallel \mathrm{W} \mid)$ numbers.

Herz[4] noted that, in the second and third terms of the recursion [equation (17)], we often need not consider the full sets L, W for the position of cuts in the rectangle (x, y) but need only consider subsets of L, W [these subsets arising from those pieces which can be cut from (x, y)]. As discussed by Christofides and Whitlock[5] such subsets are automatically generated by their dynamic-programming algorithm for producing L and W. Computationally this improvement can be applied

to the general guillotine cutting recursion [equation (17)] and to the staged cutting recursion [equations (15) and (16)].

However, our experience has been that, for any particular problem, either the recursions presented above are computationally feasible or the problem is so large that they are computationally infeasible and this improvement would not make a significant difference. Accordingly, in the computational results reported later, we have not implemented this improvement.

In the next section we describe the heuristic algorithm for general (non-staged) guillotine cutting which we can use if the optimal recursion [equation (17)] becomes computationally infeasible.

Heuristic algorithm

We said before that the optimal recursion [equation (17)] becomes computationally infeasible if $|L|$ or $|W|$ is large. In order to overcome this, we redefine L and W to ensure that $|L|$ and $|W|$ are small. We shall illustrate this redefinition with respect to L (the redefinition of W being very similar).

Let M be a limit on $|L|$ such that we require $|L| \leqslant M$. M can be determined by considering the number of operations (or storage capacity) required by equation (17) and deciding the maximum number of operations (or storage capacity) we consider computationally feasible. Then we put forward the following procedure for redefining L.

(1) Let $N = [1, 2, \ldots, m]$ (N represents the pieces to be cut from A_0).

(2) Calculate L from

$$L = \left[x \mid x = \sum_{i \in N} L_i a_i,\ 1 \leqslant x \leqslant L_0 - \min(L_j \mid j \in N),\ a_i \geqslant 0 \quad \text{and} \quad \text{integer } i \in N \right]. \tag{18}$$

(3) If $|L| \leqslant M$, then stop with L the set that we require; otherwise go to step (4).

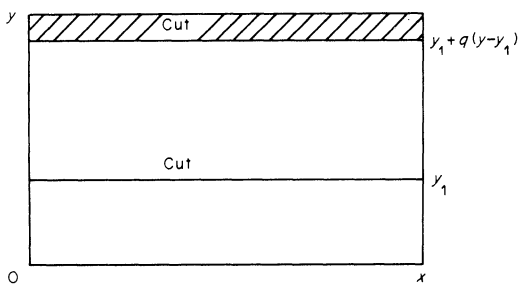

FIG. 4. *Cutting for the modified recursion.*

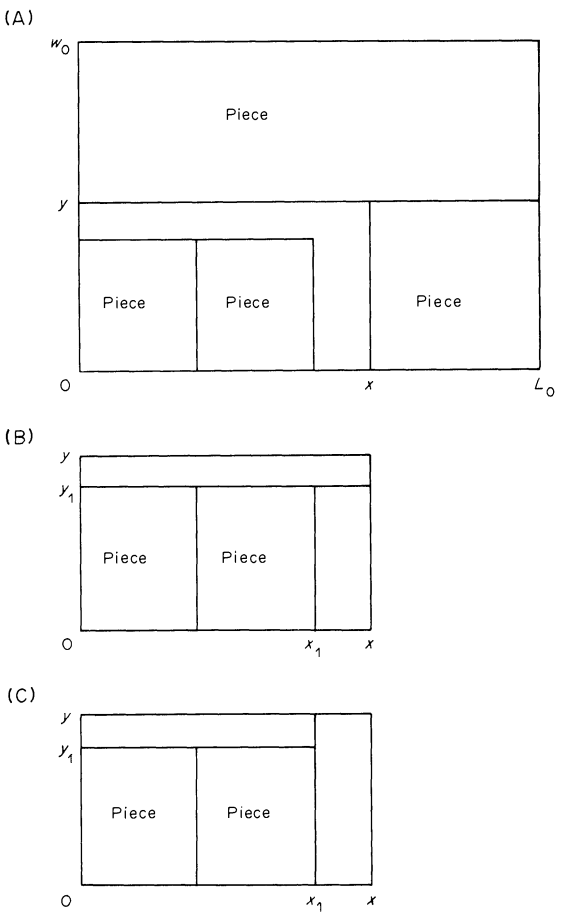

FIG. 5. *Improving the heuristic.*

(4) Define

$$L_i = \min(L_j \mid i \in N); \tag{19}$$

then set $N = N - [j]$ and go to step (2).

Essentially the above procedure removes the piece with the smallest length from N until L is the required size.

With L and W redefined as described above, we can no longer guarantee an optimal solution and we have the heuristic dynamic-programming recursion:

$$F(-, x, y) = \max[F(0, x, y); F(-, x, y_1) + F[-, x, q(y - y_1)], y_1 \in W, y_1 \leqslant y - 1;$$
$$F(-, x_1, y) + F[-, p(x - x_1), y], x_1 \in L, x_1 \leqslant x - 1] x \in L^0, y \in W^0. \tag{20}$$

Examination of early computational results from the heuristic recursion showed that we could significantly improve it at negligible computational cost. First we redefine $F(0, x, y)$ so that, instead of trimming at most one piece from (x, y), we trim as many pieces as possible. Let $\lfloor z \rfloor$ represent the largest integer less than or equal to z; then we redefine $F(0, x, y)$ by

$$F(0, x, y) = \max(0, \lfloor x/L_i \rfloor \lfloor y/W_i \rfloor v_i \mid L_i \leqslant x, W_i \leqslant y \; i = 1, \ldots, m). \tag{21}$$

Secondly, consider Figure 5(a), which shows a guillotine cutting pattern that could be produced by the heuristic recursion. From the bottom left-hand rectangle of size (x, y) we cut two pieces using the redefined $F(0, x, y)$. It is clear that, in practice, these two pieces could be cut in either of the ways shown in Figure 5(b) and (c). Both of these ways create waste rectangles out of which it may be possible to cut further pieces, and this can conveniently be checked using the redefined $F(0, x, y)$ [e.g. for Figure 5(b), the value of the pieces that we can cut from the waste rectangles is $F(0, x - x_1, y_1) + F(0, x, y - y_1)$, and for Figure 5(c), the corresponding value is $F(0, x - x_1, y) + F(0, x_1, y - y_1)$ and we can choose the best of these two values].

Although it is difficult to introduce this improvement into the heuristic recursion, it is a trivial task to introduce it into the backtracking routine which constructs the cutting pattern from $F(-, L_0, W_0)$.

COMPUTATIONAL RESULTS

The optimal staged guillotine cutting algorithm and the optimal general (non-staged) guillotine cutting algorithm were programmed in FORTRAN and run on a CDC 7600 using the FTN compiler with maximum optimization for a number of randomly generated problems.

For these problems, the length L_i of each piece was generated by sampling an integer from the uniform distribution $[L_0/4, 3L_0/4]$, with the width W_i of each piece being generated by sampling an integer from the uniform distribution $[W_0/4, 3W_0/4]$. The value v_i of each piece was set equal to the area $(L_i W_i)$ of the piece. In staged cutting we allowed trimming parallel to both the x-axis and the y-axis (i.e. $T = 4$).

Table 1 gives the results for the problems solved. In that table we give, for each problem, the size of the normalized sets L and W, together with the optimal value for general guillotine cutting

TABLE 1. *Computational results—optimal cutting*

Problem number	Number of pieces m	Rectangle size (L_0, W_0)	Normalized sets $\lvert L \rvert$	Normalized sets $\lvert W \rvert$	Optimal value $F(-, L_0, W_0)$	Total time CDC 7600 seconds	Optimal value stage n = 1	Optimal value stage n = 2	Optimal value stage n = 3	Total time CDC 7600 seconds
1	10	(250, 250)	26	7	56460	0.09	30728	56460	56460	0.27
2	20		37	50	60536	0.23	38172	60076	60536	0.47
3	30		79	40	61036	0.37	42012	60133	61036	0.64
4	50		83	82	61698	0.87	44652	61698	61698	1.40
5	10	(500, 500)	17	25	246000	0.10	171808	246000	246000	0.32
6	20		32	40	238998	0.19	175725	235058	238998	0.38
7	30		64	31	242567	0.24	165460	242567	242567	0.45
8	50		95	134	246633	1.41	175392	245758	245758	2.38
9	10	(1000, 1000)	29	9	971100	0.14	629928	971100	971100	0.31
10	20		27	53	982025	0.17	697170	982025	982025	0.39
11	30		67	107	980096	0.65	668608	974638	980096	1.17
12	50		153	122	979986	1.95	719592	977768	979986	3.29

TABLE 2. *Test problem details*

Piece i	Length L_i	Width W_i
1	365	185
2	378	200
3	410	165
4	425	148
5	425	296
6	439	116
7	464	1006
8	520	205
9	520	350
10	540	530
11	549	1413
12	549	1882
13	553	496
14	555	755
15	555	496
16	555	659
17	567	473
18	572	592
19	572	975
20	572	1175
21	572	1575
22	572	1390
23	572	1490
24	572	1590
25	572	1690
26	572	1890
27	610	625
28	660	490
29	690	447
30	949	445
31	949	478
32	970	463

$(L_0, W_0) = (3000, 3000)$
Maximize area of the pieces cut

TABLE 3. *Computational results—heuristic cutting*

M	Normalized sets $\|L\|$	$\|W\|$	Solution value $F(-, L_0, W_0)$	Solution percentage $[F(-, L_0, W_0)/(L_0 W_0)] \times 100\%$	Time CDC 7600 seconds
25	23	24	8863620	98.485	0.38
50	42	42	8863620	98.485	0.83
75	74	70	8863620	98.485	2.27
100	74	70	8863620	98.485	2.29
125	125	109	8868950	98.544	6.89
150	125	109	8868950	98.544	6.90

$F(-, L_0, W_0)$ and the optimal staged cutting values $F(n, L_0, W_0)$ for $n = 1, 2, 3$ (i.e. a maximum of three stages, where the cut direction at the first stage is parallel to the x-axis), together with the associated times in CDC 7600 seconds. It is clear from Table 1 that the algorithms presented in this paper are able to solve optimally relatively large guillotine cutting problems.

The heuristic algorithm for general (non-staged) guillotine cutting was also programmed in FORTRAN and run on a CDC 7600 using the FTN compiler with maximum optimization. To investigate this heuristic algorithm we solved the large guillotine cutting problem shown in Table 2. This problem was derived from a practical problem involving cutting a three-metre square stock rectangle, with piece dimensions being measured to millimetre accuracy.

Table 3 shows the heuristic solution value $[F(-, L_0, W_0)]$, together with the associated time in CDC 7600 seconds for varying values of M (see the previous section). For all the values of M shown, we redefined L and W so that $|L| \leqslant M$ and $|W| \leqslant M$. As we do not know the optimal solution for the problem presented in Table 2, it is difficult to ascertain how far from optimal the solution values given in Table 3 are. However, it is plain that since, for the problem we are considering, we are maximizing the area of the cut pieces, the optimal solution cannot exceed the area of the stock rectangle $(L_0 W_0)$.

Hence, to evaluate the heuristic algorithm we give in Table 3 the percentage of the stock rectangle area used by the cut pieces $\{100[F(-, L_0, W_0)/(L_0 W_0)]\%\}$ for the various values of M.

These percentages indicate that, for the problem we considered, the heuristic produces near-optimal results, even for quite small values of M.

CONCLUSIONS

In this paper we considered the unconstrained two-dimensional guillotine cutting problem and developed dynamic-programming recursions for staged cutting and general (non-staged) guillotine cutting. We showed how the use of normal patterns could lead to improved recursions. Computational results were presented both for staged cutting and for general guillotine cutting, and these indicate that the algorithms developed are capable of dealing effectively with large problems.

REFERENCES

1. J. E. Beasley (1982) An exact two-dimensional non-guillotine cutting tree search procedure. To appear in *Opns Res.*
2. P. C. Gilmore and R. E. Gomory (1965) Multistage cutting problems of two and more dimensions. *Opns Res.* **13,** 94–120.
3. P. C. Gilmore and R. E. Gomory (1966) The theory and computation of knapsack functions. *Opns Res.* **14,** 1045–1074.
4. J. C. Herz (1972) A recursive computing procedure for two-dimensional stock cutting. *I.B.M. Jl Res. Dev.* **16,** 462–469.
5. N. Christofides and C. Whitlock (1977) An algorithm for two-dimensional cutting problems. *Opns Res.* **25,** 30–44.
6. O. B. G. Madsen (1980) References concerning the cutting stock problem. IMSOR working paper, The Technical University of Denmark, Lyngby.
7. P. Y. Wang (1983) Two algorithms for constrained two-dimensional cutting stock problems. *Opns Res.* **31,** 573–586.
8. S. G. Hahn (1968) On the optimal cutting of defective sheets. *Opns Res.* **16,** 1100–1114.
9. A. Farley (1983) Trim-loss pattern rearrangement and its relevance to the flat-glass industry. *Eur. J. Opl Res.* **14,** 386–392.
10. A. Farley (1983) A note on modifying a two-dimensional trim-loss algorithm to deal with cutting restrictions. *Eur. J. Opl Res.* **14,** 393–395.
11. D. Abel, H. Dyckhoff, T. Gal and H. J. Kruse (1985) Trim loss and related problems. *Omega* **13,** 59–72.
12. M. Adamowicz and A. Albano (1976) A solution of the rectangular cutting-stock problem. *IEEE Trans. Syst. Man Cybernet.* **SMC-6,** 302–310.
13. M. J. Haims and H. Freeman (1970) A multistage solution of the template-layout problem *IEEE Trans. Syst. Sci. Cybernet.* **SSC-6,** 145–151.