Packing Rectangular Pieces—A Heuristic Approach

Bengt-Erik Bengtsson
Lund Institute of Technology, Department of Computer Science, Box 725, S-220 07 Lund, Sweden

The following two-dimensional bin packing problems are considered. (1) Find a way to pack an arbitrary collection of rectangular pieces into an open-ended, rectangular bin so as to minimize the height to which the pieces fill the bin. (2) Given rectangular sheets, i.e. bins of fixed width and height, allocate the rectangular pieces, the object being to obtain an arrangement that minimizes the number of sheets needed. Approximation algorithms are given where the solutions are achieved with heuristic search methods. Test results are presented to support the feasibility of the methods.

INTRODUCTION

A wide variety of space- and resource-usage problems can be formulated as two-dimensional bin packing tasks. Obvious industrial applications are the lay-out of templates on a stock sheet in order to minimize waste—the placement of sewing patterns onto pieces of material, the nesting of steel plates in the shipbuilding industry, and similar problems.

An interesting application is the allocation of jobs to a shared storage multiprocessor system. In this case rectangular pieces represent jobs with known space and time requirements, the horizontal dimension is memory, and the vertical dimension is time. These pieces are not allowed to rotate!

An example of a three-dimensional allocation problem is the generation of pallet loading patterns, e.g. the problem encountered when loading a large shipping container. In a commonly adopted approach this basically three-dimensional box packing is reduced to the two-dimensional problem of packing rectangles orthogonally into fixed size enclosing rectangles, thus filling the container layer by layer.

The general allocation problem may include irregular as well as rectangular shapes, and no restrictions are imposed on the orientation of the shapes. This paper discusses an algorithm to solve lay-out problems with rectangular pieces only. We will assume that the pieces may be rotated by 90°, but not by any other angle.

NESTING METHODS

During the last ten years, a number of nesting methods have been developed, ranging from manual nesting to methods completely automated by computer. In many shipyards, nesting is handled with a system which is essentially manual, although a computer is used to improve accuracy and efficiency. Descriptions of the pieces are processed and filed by the computer. A plotter produces copies of the pieces. The pieces are cut out and arranged manually into a frame which corresponds to the stock sheet. The locations are fed into the computer by terminal or with a digitizing table. The final drawing is generated by the computer on a graphic device.[1]

Some methods might be classified as semi-automatic. Nesting is carried out using an interactive graphics device. The pieces and the sheet are displayed on the screen. The operator can rotate the pieces and move them around the screen by using some input device such as a light pen or a joystick. During the nesting the computer continuously checks overlapping and other possible errors, and calculates the amount of waste.[2]

Methods for automatic nesting have been developed, but in most cases they have been marginally useful since they have proved too expensive in terms of computer time. However, in recent years a number of successful approaches have been reported.

A method based on random trials is described in Refs 3 and 4. In Refs 5 and 6, a two-stage solution is suggested. The first stage is concerned with producing 'optimal' rectangular enclosures, called modules, for the shapes and for various clusterings of the shapes, and the second stage finds 'optimal' layouts of the modules on the rectangular sheets. In Ref. 7 an approach is given where the solution is achieved with a heuristic method typical of the artificial intelligence discipline. The allocation problem is transformed into a breadth-first search and traversal of a directed graph, where each state corresponds to the allocation of one piece according to a placement policy.

Restricting the permissible nestings to those attainable with one or more guillotine cuts (a cut goes from one edge to the opposite one) drastically reduces the number of combinations open. An extremely efficient algorithm for this special case is described in Ref. 8.

It has been argued that in industrial applications the nesting process includes a large number of objectives, along with the minimization of waste, which cannot be fully automated.[9,10] Automated nesting should thus be combined with an interactive, visual evaluation.

THE HEURISTIC APPROACH

In the first place we shall abandon the quest for an optimal solution. We are looking for an algorithm for large instances of the layout problem and it is desirable to find a method which yields a solution 'close to' the optimal, using a reasonable amount of computing time.

CCC-0010-4620/82/0025-0353 $02.50

It may be that this solution is in fact optimal—a fact we shall only rarely be able to establish.

What we would like is heuristics to replace the huge set of possible layouts by a small subset containing feasible layouts with small waste. The layouts in this subset are then to be scanned and evaluated.

We are immediately faced with a number of questions.

Question (1). How should this subset be chosen? The choice is necessarily intuitive in nature, as we are unable to characterize the optimum or near-optimum layouts.

Question (2). How do we enumerate the layouts belonging to this set? Clearly we have to impose an implicit or explicit ordering of the layouts, if we want to scan them in sequence.

Question (3). The algorithm should include criteria to test if a layout being formed has any chance of beating the best one found so far. The overall computing time is reduced if some layouts can be excluded at an early stage.

These considerations apply to a variety of heuristic methods. However, the mechanisms that guide the search must be derived from the problem environment. Heuristic schemes are very dependent on the particular problem being solved.

PACKING INTO AN OPEN-ENDED RECTANGLE

In this section we consider an open-ended rectangular sheet of width W, and a collection of n rectangular pieces defined by pairs (l_i, l_{i+1}), $i = 1,3,\ldots, 2n-1$, corresponding to the horizontal and vertical dimensions.

We are concerned with the following problem. Find a way to pack the pieces into the sheet so as to minimize the total length, L, of the space occupied by any of the pieces. The pieces are allowed to rotate 90° but must not overlap. Figure 1 illustrates a possible arrangement of eight pieces. The shaded areas are unused. The pieces were positioned in the order shown by the encircled numbers. A coordinate system is introduced, for convenience, with x-axis along the bottom edge and y-axis along the left edge of the sheet.

The following rather natural packing scheme was chosen as a basis for a heuristic algorithm. Out of all possible arrangements, select those where the pieces form 'piles without spurs'. In Fig. 1 the pieces Nos 1, 2, 3 and 4 form such a pile without overhanging parts or spurs, as the horizontal lengths decrease. Note that after

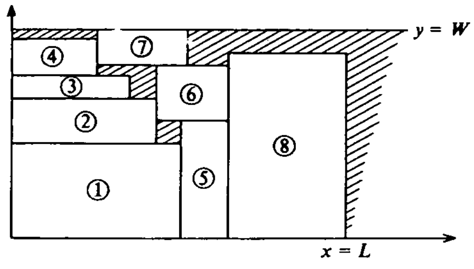

Figure 1. A possible arrangement of eight pieces.

addition of pieces Nos 5, 6 and 7 to the pile, this condition is still fulfilled although these pieces are of increasing horizontal length. The sheet is filled from left to right in sections. This answers question (1) of the preceding section, at least preliminarily and informally. As for question (2) it was decided to generate and compare a number of combinations by starting with different pieces at the bottom of the sections.

Now for a more detailed specification of the method. In what follows, every piece to be packed, as well as the sheet, will be principally oriented, so that either its length or its width direction is horizontal. A few definitions will make it easier to describe the algorithm.

A *section* is an ordered set of pieces such that every horizontal line not coincident with one of the edges of the rectangles passes through at most one rectangle. (In sections produced by the algorithm consecutive pieces touch along a horizontal edge, except in rare cases.)

A *pile* is an ordered set of sections such that every piece touches either the y-axis or at least one piece belonging to one of the preceding sections (not necessarily the immediate predecessor) along a vertical edge.

Now, let the pieces be numbered sequentially starting with the pieces of section No. 1 (giving the encircled numbers in Fig. 1). Let the coordinates of the upper right corner of piece No. k be denoted by (x_k, y_k).

A section is *well formed* if the values of x_k form a nonincreasing sequence. A pile is well formed if its sections are all well formed.

The pile in Fig. 1 is well formed as the following three conditions hold true: $x_1 \geq x_2 \geq x_3 \geq x_4$ and $x_5 \geq x_6 \geq x_7$ and only one piece in section 3.

The algorithm:

```
procedure pack;
if good-hope then
begin
  piece := first-piece-in-size;
  while piece ≠ nil do
  begin
    if piece is free then
    begin
      add a section with piece as base;
      pack; (*the procedure is called recursively*)
      check for minimum;
      delete last section
    end;
    piece := next-piece-in-size
  end
end
```

The function *good-hope* makes a simplified prognosis. Suppose that it is possible to pack the remaining pieces with no further waste. Will it then be possible to beat the best solution found so far? If so, *good-hope* is true.

First-piece-in-size refers to the piece with one side equal to max (l_i), $i = 1,2,\ldots, 2n$; *next-piece-in-size* to the piece with next value in decreasing order. As every piece occurs twice in this enumeration both orientations will be tried.

A section is built starting with the selected piece as base, by repeatedly adding the piece with the largest possible length. In Fig. 1, piece No. 6 was selected among the not yet allocated pieces, subject to the conditions $l_i \leq x_5 - x_2$ and $l_{i+1} \leq W - y_5$ (or vice

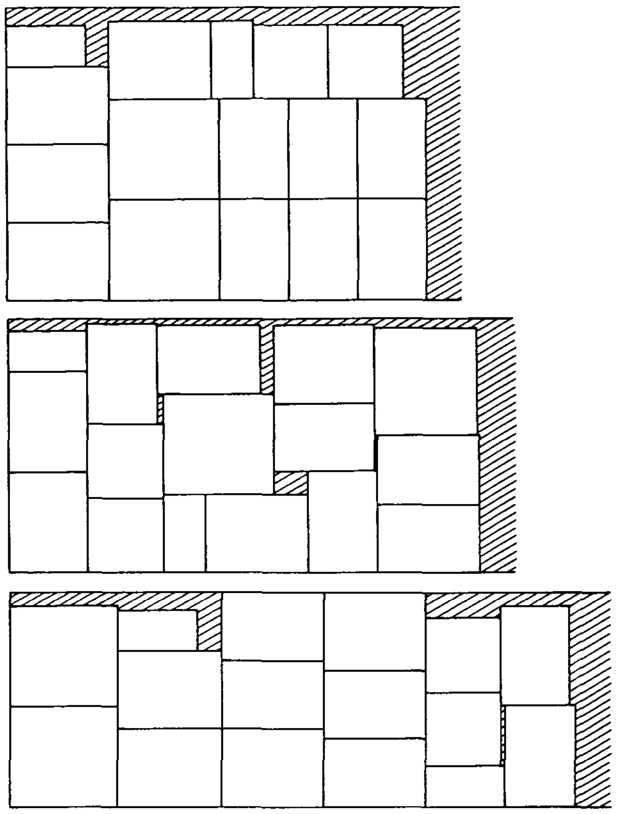

Figure 2. Packing into open-ended rectangles—three different widths.

versa). In general, we favour the piece with maximum horizontal length. When selecting the topmost piece it is however regarded as favourable to come close to the second limit.

The check for minimum includes verifying that all pieces have been allocated. This is necessary as a return from the recursive call of the procedure pack is due to (a) unacceptable waste in a partial solution, or (b) no pieces remain, or (c) all combinations have been evaluated.

The algorithm was programmed in Algol. Figure 2 provides examples of layouts produced by the program. A set of 16 pieces were arranged on sheets of different widths. The computing times were 1.1, 1.1, and 5.9s on a Univac 1100/80.

PACKING INTO RECTANGLES OF FIXED SIZE

The described packing procedure has been slightly modified and extended to handle the following problem. Given rectangular sheets of fixed length and width and a set of rectangular pieces, find an arrangement of the pieces that minimizes the number of sheets needed. If this number is S, find an arrangement that makes maximum use of $S - 1$ sheets, thus leaving a connected area unused on the last sheet.

The same technique as in the one sheet problem is adopted to generate a sequence of acceptable solutions. Backtracking takes place when it is no longer possible to add sections within the fixed sheet length.

The algorithm

Initial allocation. Carry out a rough distribution of the pieces into a sufficient number of sheets. Backtracking is dispensed with, to avoid a possible concentration of the small and easily manageable pieces into a few sheets.

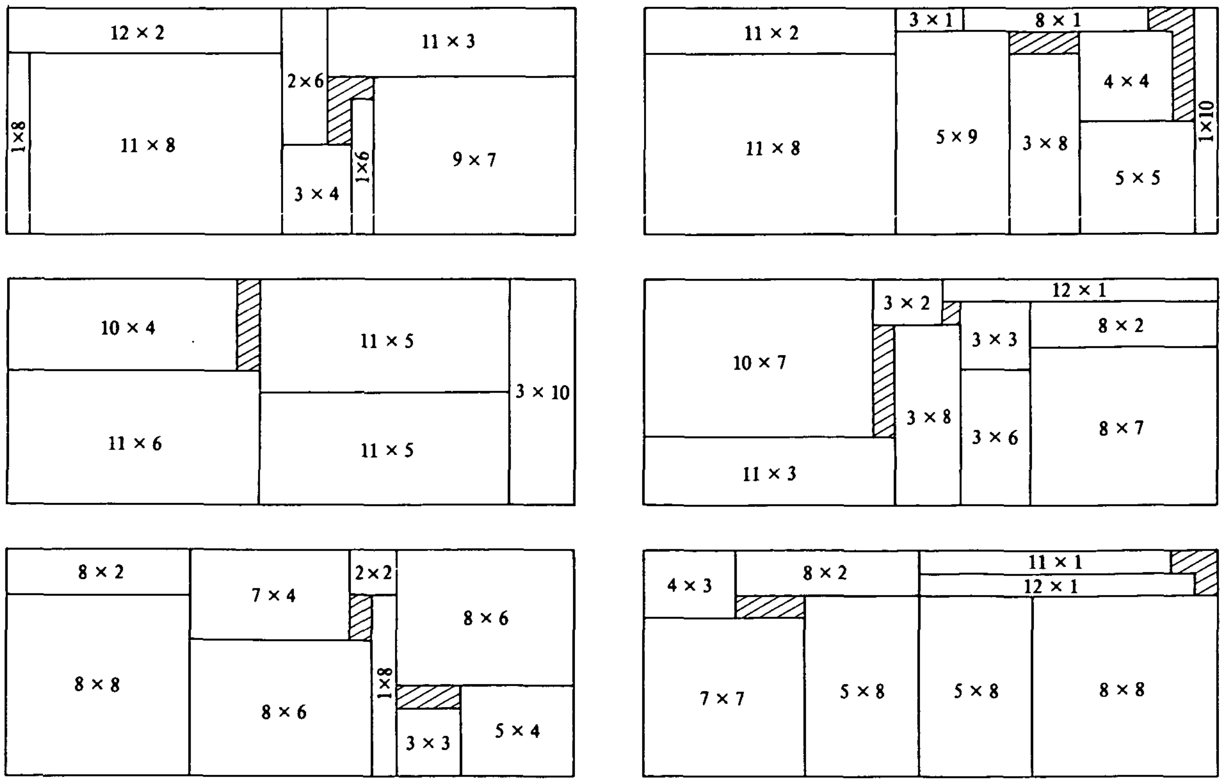

Figure 3. Packing into fixed size rectangles.

Create pool. Discard the sheet with maximum waste. Transfer the pieces to a pool. These pieces will be used in the efforts to improve utilization of the remaining sheets. Mark these sheets as active (unfinished).

Repeat until all sheets are inactive. Among the active sheets select the one with maximum waste. Look for an improved arrangement, using the packing algorithm upon pieces from this sheet and pieces from the pool. The outcome is one of the following.

(a) No improvement was achieved. The sheet is marked as inactive.
(b) The layout was improved, but the sheet is still the worst one. The new arrangement is accepted, the sheet is marked as inactive, the other sheets are all marked as active.
(c) An improved layout is found, the sheet is no longer the worst one. As soon as such an arrangement is found, the search process is interrupted. This is a precaution to prevent the packing from ending up with all the small and easily allocated pieces collected on the same sheet. The new arrangement is accepted, the sheets are all marked as active.

In cases (b) and (c) the pool might be emptied. If so 'create pool' is invoked again. Figure 3 shows a layout obtained by the algorithm. A set of rectangular pieces with integer lengths ranging from 1 to 12 and widths ranging from 1 to 8 were chosen randomly. Figure 3 shows an arrangement of 48 pieces on to six sheets of size 25 × 10. Two pieces, of sizes 11 × 3 and 8 × 1, remain in the pool. The computer used 13.8s to find this layout and then another 20s with the algorithm in futile efforts to improve it. The computation time was similar for other sets of pieces produced in the same way. Overall sheet utilization was typically 95% to 98%.

EXPERIMENTAL RESULTS

The present work originates from inquiries for automatic nesting methods made by a manufacturer of excavators

Table 1. Pieces for test problems

1–40	41–80	81–120	121–160	161–200
8 * 6	10 * 7	5 * 6	11 * 4	8 * 3
11 * 4	5 * 4	11 * 1	8 * 1	5 * 5
10 * 3	3 * 2	1 * 1	10 * 5	3 * 4
4 * 7	7 * 7	7 * 7	11 * 8	3 * 6
12 * 6	11 * 8	3 * 8	1 * 6	4 * 3
11 * 6	10 * 5	10 * 8	12 * 6	3 * 8
12 * 8	4 * 7	4 * 7	1 * 8	8 * 2
3 * 6	5 * 4	7 * 6	6 * 4	8 * 1
4 * 3	2 * 3	9 * 5	1 * 8	3 * 4
3 * 8	3 * 3	5 * 2	2 * 4	3 * 5
9 * 1	8 * 3	7 * 3	1 * 6	6 * 8
1 * 3	11 * 4	11 * 6	1 * 5	10 * 6
8 * 6	8 * 1	9 * 7	6 * 1	2 * 3
11 * 3	8 * 7	7 * 6	2 * 1	6 * 3
11 * 1	6 * 5	9 * 6	9 * 5	7 * 7
9 * 8	8 * 5	4 * 2	10 * 8	7 * 8
9 * 1	4 * 8	4 * 6	10 * 1	9 * 3
11 * 5	8 * 8	3 * 1	1 * 4	12 * 4
1 * 7	7 * 2	12 * 4	7 * 8	9 * 1
7 * 8	6 * 2	4 * 2	5 * 4	12 * 3
10 * 3	9 * 2	9 * 5	11 * 3	8 * 5
2 * 8	7 * 6	6 * 7	4 * 7	11 * 2
8 * 3	1 * 5	2 * 1	1 * 5	3 * 3
7 * 3	5 * 2	6 * 5	1 * 3	1 * 1
8 * 4	11 * 4	11 * 4	9 * 5	4 * 7
2 * 7	11 * 3	4 * 2	1 * 1	4 * 5
12 * 7	3 * 1	8 * 6	7 * 7	11 * 8
1 * 2	9 * 2	1 * 6	3 * 1	2 * 6
3 * 8	11 * 7	5 * 5	7 * 4	2 * 4
1 * 1	10 * 4	5 * 5	3 * 8	8 * 8
12 * 7	5 * 4	10 * 5	5 * 2	7 * 2
3 * 4	11 * 2	11 * 8	11 * 3	7 * 1
8 * 4	7 * 3	2 * 6	12 * 1	10 * 8
11 * 5	12 * 6	1 * 5	10 * 7	12 * 8
5 * 6	11 * 5	6 * 3	2 * 3	6 * 6
8 * 8	2 * 6	11 * 5	8 * 1	1 * 1
4 * 1	10 * 4	9 * 6	1 * 6	5 * 5
10 * 3	10 * 4	8 * 4	8 * 6	10 * 5
12 * 7	1 * 3	12 * 7	2 * 5	11 * 7
12 * 3	2 * 4	8 * 3	9 * 1	2 * 4

Table 2. Sheet size 25 × 10

Number of pieces	20	40	60	80	100
Area of pieces	741	1420	2090	2673	3330
Time limit = 0.5s:					
Number of sheets	3	5	8	10	13
Waste	64	34	70	26	29
Waste rate	8.5%	2.7%	3.5%	1.0%	0.9%
CPU time	1.1s	6.4s	6.3s	16.6s	24.0s
Left in pool	1	5	5	6	3
Area of these	55	204	160	199	109
Time limit = 1.0s:					
Number of sheets	3	5	8	10	13
Waste	64	34	33	7	32
Waste rate	8.5%	2.7%	1.6%	0.3%	1.0%
CPU time	1.6s	11.3s	25.1s	30.7s	23.8s
Left in pool	1	5	3	6	2
Area of these	55	204	123	180	112

Table 3. Sheet size 40 × 25

Number of pieces	40	80	120	160	200
Area of pieces	1420	2673	4027	5008	6217
Time limit = 0.5s:					
Number of sheets	1	2	4	5	6
Waste	121	69	131	124	72
Waste rate	12.1%	3.4%	3.3%	2.5%	1.2%
CPU time	0.9s	5.4s	14.1s	20.1s	49.6s
Left in pool	23	34	12	16	14
Area of these	541	742	158	132	289
Time limit = 1.0s:					
Number of sheets	1	2	4	5	6
Waste	121	85	113	100	97
Waste rate	12.1%	4.3%	2.8%	2.0%	1.6%
CPU time	1.4s	8.3s	15.9s	67.5s	39.5s
Left in pool	23	25	2	18	11
Area of these	541	758	140	108	314

and power shovels, so sequences of test problems were generated with this application in mind.

In this manufacturing process, and certainly in many other industrial applications, the pieces are relatively large compared to the sheet of raw material. Thus one sheet can hold a small number of pieces only, say between 5 and 20. A production plan may cover a few hundred of pieces of different sizes and shapes. As a great many of the pieces are roughly rectangular it was decided that a first stage should be to develop a program for rectangular shapes.

A set of 200 pieces was generated randomly according to the assignments:

```
length := entier(12 * random + 1);
width := entier(8 * random + 1)
```

where the value of random is drawn from a uniform distribution in the range (0, 1). (Table 1.) Two sheet sizes were considered: 25 * 10 and 40 * 25. A time limit was introduced: if for the current sheet no improved solution is found within the specified time, this sheet is abandoned for the present and marked as inactive. The experiments were carried out with two different time limits, 0.5s and 1.0s. The results are summarized in Tables 2 and 3.

The waste rate decreases as the number of pieces taken into account increases. In most cases the algorithm is able to improve the solution further at the expense of increasing computing time.

In the case of 40 pieces, sheet size 40 * 25, the time limit is somewhat inappropriate as the pieces are packed into one single sheet, so the computations were resumed without the time limit. The 12% solution was gradually improved: 5% waste was reached in 4s and 1% in 56s.

It seems that no previous papers deal with this precise problem, so direct comparisons cannot be made between this algorithm and other approaches. Results obtained by Albano and Orsini[8] are remarkably good: 1 or 2% waste to the cost of 0.1s–0.5s on an IBM 370/168. However their results apply to the nesting of a large number, some hundred per sheet, of small rectangular pieces considered in groups or 'strips'. The algorithm relies on dynamic programming techniques and cannot readily be generalized to irregular shapes.

In Ref. 7, Albano and Sapuppo adapt a heuristic search strategy to problems involving 24 or 30 irregular shapes, to be nested into one sheet. Computing times range from 8s to 160s and the waste rates from 17% to 27%.

CONCLUSIONS

A heuristic approach to the two-dimensional allocation problem has been described. The main advantage of the method is in its simplicity and small memory demands. Experiments indicate that the algorithm would be an effective tool in designing layouts for problems that involve rectangular pieces, to be allocated upon rectangular sheets. A wide range of allocation problems is of this type.

Extensions of the problem where irregular pieces occur could be attacked with the same general packing strategy. As an alternative, the solution may be approached in two stages, as suggested in Ref. 5. In the first stage the pieces are encased in rectangular modules either singly or in combination with other pieces. Then these modules are used in the second stage to produce layouts on the rectangular sheets.

Work still needs to be done to test the effectiveness of the approach. This work should however be directed towards specific industrial applications.

REFERENCES

1. J. Derse and K. Halbauer, Erstellen eines schachtelplans für schneidmaschinen mit dem digitalisiergerät. *Maschinenmarkt* **85** (No. 75), 1468–1470 (September 1979).
2. Y. Ikeda, The Pansy, an advanced interactive parts nesting system, in *Computer Applications in the Automation of Shipyard Operation and Ship Design*, ed. by C. Kuo, K. J. MacCallum and T. J. Williams, Vol. 3, pp. 313–321. IFIP, North-Holland (1979).
3. H.-P. Niederhausen, Programm zum automatischen herstellen eines schneidplans für beliebig gestaltete werkstücke durch elektronische datenverarbeitung. *Schweissen und Schneiden* **29** (No. 3), 103–105 (1977).
4. D. Böhme and A. Graham, Practical experiences with semi-automatic and automatic partnesting methods, in *Computer Applications in the Automation of Shipyard Operation and Ship Design*, ed. by C. Kuo, K. J. MacCallum and T. J. Williams, Vol. 3, pp. 213–220. IFIP, North-Holland (1979).
5. M. Adamowicz and A. Albano, Nesting two-dimensional shapes in rectangular modules. *Computer Aided Design* **8** (No. 1), 27–33 (January 1976).
6. M. Adamowicz and A. Albano, A solution of the rectangular cutting-stock problem. *IEEE Transactions on Systems, Man and Cybernetics* **SMC-6** (No. 4), 302–310 (April 1976).
7. A. Albano and G. Sapuppo, Optimal allocation of two-dimensional irregular shapes using heuristic search methods. *IEEE Transactions on Systems, Man and Cybernetics* **SMC-10** (No. 5), 242–248 (May 1980).
8. A. Albano and R. Orsini, A heuristic solution of the rectangular cutting stock problem. *The Computer Journal* **23** (No. 4), 338–343 (November 1980).
9. K. J. MacCallum and I. S. Smith, The representation and manipulation of two-dimensional shapes for nesting, in *Third International Conference and Exhibition on Computers in Engineering and Building Design*, pp. 600–611. IPC Science and Technology Press (1978).
10. B. Sperling, Nesting is more than a layout problem, in *Computer Applications in the Automation of Shipyard and Ship Design*, ed. by C. Kuo, K. J. MacCallum and T. J. Williams, Vol. 3, pp. 287–294. IFIP, North-Holland (1979).

Received June 1981