AN ALGORITHM FOR A CLASS OF LOADING PROBLEMS

Ming S. Hung

College of Business Administration
Cleveland State University
Cleveland, Ohio

J. Randall Brown

College of Business Administration
Kent State University
Kent, Ohio

ABSTRACT

The loading problem we consider is to assign a set of discrete objects, each having a weight, to a set of boxes, each of which has a capacity limit, in such a way that every object is assigned to a box and the number of boxes used is minimized.

A characterization of the assignments is offered and used to develop a set of rules for generating nonredundant assignments. The rules are incorporated into an implicit enumeration algorithm. The algorithm is tested against a very good heuristic. Computational experience shows that the algorithm is highly efficient, solving problems of up to 3600 0-1 variables in a CPU second.

1. INTRODUCTION

One form of the loading problem [2] is to assign a set of m objects, each having a certain weight w_i, $i = 1, \cdots, m$, to a number of boxes, each of which has a weight capacity limit c_j, $j = 1, \cdots, n$, in such a way that every object is assigned to a box and the number of boxes used is minimized.

There are many applications of the loading problem. Eilon, et al. [3] considered the problem of determining the number of vehicles to carry a given consignment to a destination. It can be used to cut rectangles from larger rectangular sheets [5] and to schedule jobs of given duration on parallel machines [7]. Johnson [10] called it a "bin packing" problem for assigning data files of given lengths to disc tracks.

Let

$x_{ij} = 1$ if object i is assigned box j, 0 otherwise.

$y_j = 1$ if box j is used, 0 otherwise.

The loading problem can be written as the 0-1 linear program:

(p) $$\text{Minimize } z = \sum_{j=1}^{n} y_j$$

subject to

(1) $$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, \cdots, m,$$

(2) $$\sum_{i=1}^{m} w_i x_{ij} \leqslant c_j y_j \qquad j = 1, \cdots, n,$$

(3) $$x_{ij} = 0 \text{ or } 1 \text{ for all } i, j.$$

Other forms of the loading problem have been suggested by Eilon and Christofides [2]. One form is to maximize the total value of the objects that can be assigned to boxes. This has been investigated by Ingargiola and Korsh [9]. Another form is to minimize the unused space in the boxes used. If all box sizes are equal, then this problem is equivalent to our problem (p).

A solution to the loading problem (p) entails two interrelated decisions: which boxes to use and how to assign the objects to the boxes. The second decision is more difficult because of the massive number of combinations that need to be investigated. This paper offers a characterization of such assignments. The characterization has been shown to be very effective in reducing the effort of searching for an optimal solution to the loading problem.

With respect to other solution methods for the loading problem, Eilon and Christofides [2] proposed an implicit enumeration algorithm for problems in which the box capacities are all equal. They also proposed a heuristic which, according to the computational experience conducted for this research, is very good and efficient. A more detailed discussion of the heuristic is given in Section 4. Johnson [10] developed four heuristic algorithms for the bin-packing problem. The heuristics are similar in nature to that of Eilon and Christofides; therefore, their quality should be comparable.

The algorithm presented in this paper is based upon the characterization of the assignments. Computational experience in Section 4 shows that the algorithm, which guarantees optimal solutions, is competitive with Eilon and Christofides' heuristic in terms of solution time.

2. CHARACTERIZING ASSIGNMENT MATRICES

Let $A = [x_{ij}]$ be an m × n matrix of zeros and ones. The rows of A correspond to objects and the columns correspond to boxes. $x_{ij} = 1$ means object i is assigned to box j. Therefore, no row in A contains more than one 1. A will be called an assignment if every row of A has exactly one 1. If some rows are zero rows (null rows), then A is called a partial assignment. (For clarity, object i is denoted by a_i and box j is denoted by b_j in this section.)

For a clearer understanding of the following development, suppose five objects, $a_1, a_2, \cdots, a_5$ are to be assigned to four boxes, $b_1, \cdots, b_4$. Further suppose there are two assignments as shown in Fig. 1.

Suppose objects a_1, a_2, and a_3 have the same weight, i.e., $w_1 = w_2 = w_3$, and boxes b_3 and b_4 have the same capacity, $c_3 = c_4$. Then A_1 and A_2 are essentially the same assignments.

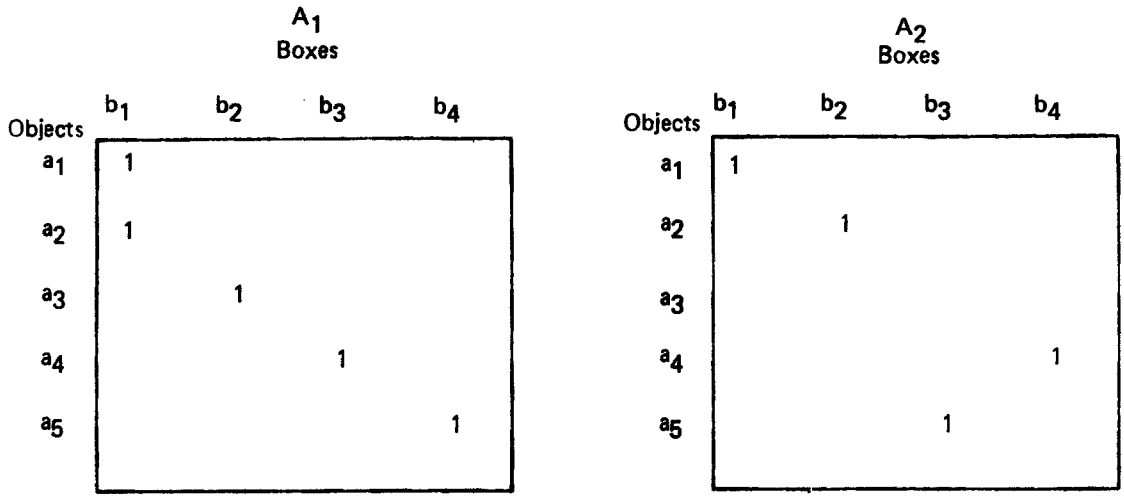

FIGURE 1. Two assignments

First, boxes b_1 and b_2 contain essentially the same objects in both A_1 and A_2 except for the indices of the objects. Second, objects a_4 and a_5 are individually assigned to essentially the same boxes because both boxes have the same capacity. Hereafter we shall refer to two assignments that are essentially the same as "equivalent assignments," and all equivalent assignments but one are "redundant." An assignment that is not equivalent to any other assignment will be called "nonredundant." These definitions are necessary because in an enumeration scheme, optimality of a solution is guaranteed only after all nonredundant solutions are considered, explicitly or implicitly.

To identify the equivalent assignments in general, let us suppose there is an equivalence relation "ρ" (e.g., same weights) among the objects and an equivalence relation "σ" (e.g., same capacities) among the boxes. If a $a_i \rho a_k$, then exchanging rows i and k in the assignment matrix yields an equivalent assignment; so does exchanging columns j and k if $b_j \sigma b_k$. To aid our identification, partition the assignment matrix by blocking together rows equivalent under ρ and columns equivalent under σ. For example, the two matrices in Fig. 1 are blocked and shown in Fig. 2.

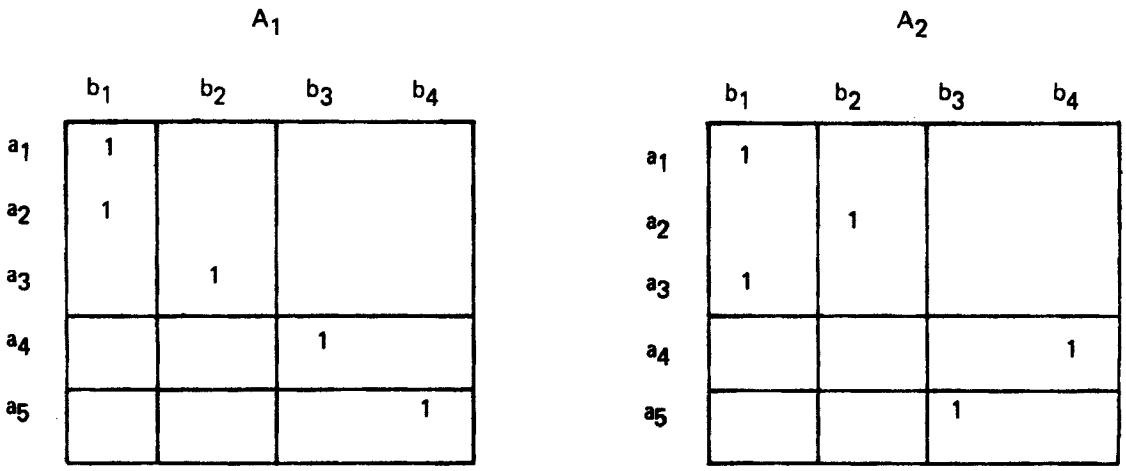

FIGURE 2. Block form of two assignments

An assignment matrix is then transformed into "normal form" by the following steps:

(a) Permute the rows within each row-equivalent block so as to put them in inverse lexicographic order; e.g., (0,1,0,0) precedes (0,0,1,0), which precedes (0,0,0,1).

(b) Permute the columns within each column-equivalent block so as to put them in inverse lexicographic order.

As for our two examples, A_1 is already in normal form, whereas the normal form A_2 is shown in Fig. 3.

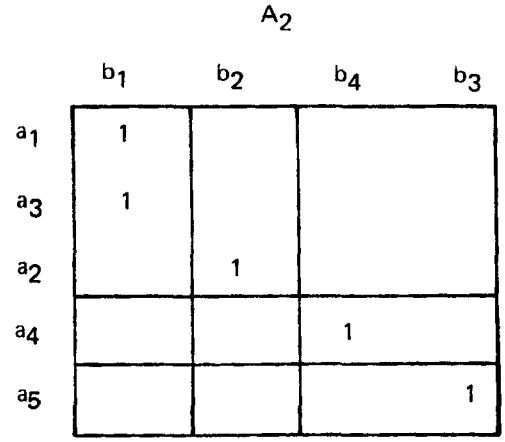

FIGURE 3. Normal form of assignment A_2

It can be seen that A_2 has the same normal form as A_1. Hence, a formal remark:

REMARK: Two assignments are equivalent if their assignment matrices have identical normal forms.

If two partial assignment matrices have the same normal form, then we also assert that for every completion of one assignment, there is an equivalent completion of the other. Thus, in order to avoid generating redundant assingments in an implicit enumeration scheme, one should branch in such a way that every partial assignment is always in normal form.

The normal forms of two equivalent assignments are not necessarily identical because the columns in the same equivalence block may contain an unequal number of 1's among the rows of an equivalence block. For example, assume that b_1 and b_2 in A_1 and A_2 (Fig. 1) are in the same column block and object a_3 is assigned to b_2 in A_2. All other assumptions about A_1 and A_2 remain the same as before. Then the normal form of A_1 and A_2 will be different even though the assignments are equivalent.

The following rules, which were developed by Brown [1], ensure that partial assignments are in normal form and that the normal form is unique. To achieve both purposes, a concept of relatively equivalent columns (boxes) is needed.

DEFINITION: Boxes b_j and b_k (columns j and k) are relatively equivalent if and only if b_j and b_k are equivalent under σ and, in every completely assigned row (object) equivalence block, columns j and k have an equal number of 1's.

A row block is completely assigned if every row has been given a 1. Thus, at the beginning, when no object has been assigned, all equivalent columns (boxes) are also relatively equivalent. Once a row block is completely assigned, some equivalent columns may become relatively unequivalent and remain so for all following partial assignments.

R1: (Selection of branching object.) Object a_i is to be considered for assignment only if (a) object a_{i-1} has been assigned, or (b) i is the smallest index of any object in its equivalence block.

R2: (Selection of branching box.)

1. Object a_i can be assigned to box b_j (i.e., x_{ij} can be fixed to one) if either (a) a_{i-1} was assigned to a box whose index is not greater than j, or (b) i is the smallest index in its equivalence row block. (This rule preserves the lexicographic ordering of rows.)

2. Object a_i can be assigned to box b_j if either (a) column $j-1$ of A is not empty, or (b) j is the smallest index in its relative column equivalence block. (This rule preserves the lexicographic ordering of columns.)

R3: Object a_i can be assigned to box b_j only if the sum of the elements of the row block of a_i in relatively equivalent column $j-1$ is at least one greater than that in column j. (This rule ensures the uniqueness of the normal form and is to be used in conjunction with R1 and R2.)

The above characterization of assignment matrices and the rules for avoiding redundant assignments can be used for any problem involving allocation of a set of discrete entities to another set. Generalized assignment problems [11] and 0-1 multiple knapsack problems [9] belong to this category.

3. THE ALGORITHM

The algorithm presented here is basically an implicit enumeration method whose branching strategy is based on Rules R1-R3. The bounding aspect of the algorithm is adapted from a lower bound suggested by Eilon and Christofides [2].

The lower bound on the objective function value of the loading problem is the fewest possible number of boxes that can contain all the objects. Therefore, it can be computed as follows, where the boxes are indexed in descending order of their capacities:

$$\underline{z}_1 = \min\left\{k : \sum_{j=1}^{k} c_j \geqslant \sum_{i=1}^{m} w_i\right\}. \tag{4}$$

$\underline{z}_1$, the initial lower bound, provides a termination rule for the solutions. If any solution uses $\underline{z}_1$ boxes then the solution procedure is stopped. This bound was found to be very effective in the extensive computational experience described in Section 4.

Before the start of the algorithm, the boxes are indexed in descending order of their capacities and the objects are indexed in descending order of their weights. Eilon and Christofides [2] used an ascending order for the boxes but our experience showed that this often leads to inferior solutions. On the ordering of objects, Johnson [10] and Golden [6] also showed that a bin-packing heuristic which uses a descending ordering performs better than other heuristics which use different orderings.

Given the list of objects and the list of boxes, the algorithm determines for each unassigned object those boxes to which it can be assigned. To determine the candidate boxes for object i, rules R1-R3 for avoiding redundant assignments are used. We also make sure that the unused capacity of each box considered is large enough for object i. Once the candidate list is set up, object i is then assigned to the lowest indexed box in the list. This assignment phase continues until all objects are assigned and a feasible solution is found.

During the course of assigning objects to boxes, some boxes may become "full." A box is full if its unused capacity is too small to accomodate any more objects. A full box and its contents can then be removed from their respective lists and a new lower bound can be computed according to (4). If, after object $i-1$ is assigned to a box, the box becomes full, then the lower bound is recomputed and denoted by $\underline{z}_i$. Of course, if the box is not full, then $\underline{z}_1 = \underline{z}_{i-1}$ for $i \geqslant 2$.

After a feasible solution is found, with the number of used boxes denoted by $\bar{z}$, the algorithm immediately backtracks to the lowest indexed object i whose $\underline{z}_i = \bar{z}$, and reassigns the object to another candidate box.

To be specific, we first define the notation used in the algorithm.

$\bar{z}$ = upper bound on the objective function value; it is initialized to ∞.

$T_j = c_j - \sum_{i=1}^{m} w_i x_{ij}$, the unused capacity of box j

U_i = Set of box indices to which object i can be assigned. U_i is determined by rules R1-R3, the condition that for every $j \in U_1$, $w_i \leqslant T_j$, and the condition that the addition of box j does not make the total number of used boxes exceed $(\bar{z} - 1)$.

$\underline{z}_i$ = lower bound after object i-1 has been assigned.

The algorithm has the following steps:

STEP 0 (Preparation): Put the boxes in descending order of their capacities (c_j), and the objects in descending order of their weights (w_i). Block boxes and object, respectively, according to their capacities and weights. Find z_1. Set $\bar{z} = \infty$ and $i = 1$.

STEP 1 (Screening): Determine U_i. If $U_i = \phi$, go to step 4.

STEP 2 (Assignment): Assign object i to the first box in U_i. Remove this box j from U_i and adjust its T_j. Compute the bound $\underline{z}_{i+1}$. If $\underline{z} \geqslant \bar{z}$, go to step 4.

STEP 3 (Forward Branching): Increment i by 1. If i does not exceed m, go to step 1. Otherwise, a feasible solution is found. Update $\bar{z}$. If $\bar{z} = \underline{z}_1$, stop because the optimal solution is found. Otherwise, find the smallest index i whose $\underline{z}_i = \bar{z}$ or which was assigned to the $\bar{z}$th box. Go to step 4.

STEP 4 (Backtracking): Decrease i by 1. If $i \geqslant 1$, go to step 2. Otherwise, the last $\bar{z}$ is the optimal value. If $\bar{z} = \infty$, the problem has no feasible solution.

4. COMPUTATIONAL EXPERIENCE

The algorithm was programmed in FORTRAN IV-G and run on a IBM 370-145. Three types of problems were randomly generated.

Type-1 problems have equal box capacities, which are equal to the maximum object weight plus 100, and the object weights are uniformly distributed between 20 and 120 in multiples of 10. Therefore, all columns in the assignment matrix are in one block and there may be as many as 10 row blocks.

Type-2 problems have equal box capacities and the object weights are uniformly distributed between 20 and 120. Therefore, there is one column block and there may be as many as 100 row blocks.

Type-3 problems have box capacities uniformly distributed between 100 and 200 in multiples of 10 and the object weights have the same distribution as Type-1 problems. Therefore there may be as many as 10 column blocks and 10 row blocks.

A total of 165 random problems of various types and various sizes were generated (see Table 1). Each problem was solved by both the algorithm and Eilon and Christofides' heuristic. Both solutions were put into subroutines and run back to back, so as to minimize the discrepancies that usually exist in computer clocking mechanisms.

TABLE 1 — *Computational Experience*

Problem		No. of Problems	Algorithm Time (in seconds)			Heuristic			
						Time (in seconds)			No. of Suboptimal Solutions
Problem Type	Size m × n		High	Low	Median	High	Low	Median	
1	30 × 30	15	0.30	0.22	0.24	0.37	0.06	0.06	2
1	40 × 40	15	1.22	0.39	0.39	0.74	0.09	0.09	2
1	50 × 50	15	0.79	0.60	0.60	1.28	0.14	0.14	2
1	60 × 60	15	1.67	0.86	0.93	2.46	0.19	0.21	2
2	30 × 30	15	450.26	0.21	0.25	0.81	0.06	0.58	3
2	40 × 40	15	0.61	0.40	0.41	1.83	0.12	1.32	4
2	50 × 50	15	46.07	0.56	0.60	2.51	0.16	1.22	3
3	30 × 30	15	0.28	0.24	0.26	0.08	0.06	0.06	0
3	40 × 40	15	0.50	0.43	0.43	0.12	0.10	0.10	0
3	50 × 50	15	0.73	0.67	0.71	0.17	0.15	0.16	0
3	60 × 60	15	1.20	1.01	1.06	0.27	0.22	0.24	0

Eilon and Christofides' heuristic [2] is an r-pass algorithm. After the boxes and the objects are numbered in the same orders as in our algorithm, it successively assigns an object to the lowest indexed box capable of accomodating it. When all the objects are assigned, one pass is complete. Then an object, which can be arbitrarily chosen, is reassigned to a box to which it was not previously assigned. The assignment of the other objects is then continued. This reassigment may be repeated up to r-1 times. In the program, when the first solution of the heuristic yields a z value greater then $\underline{z}_1$, r is set to $[\underline{z}_1/3]$, where [ ] denotes the largest integer part.

The computational experience reported in Table 1 reveals:

- The algorithm is very efficient. For the 165 test problems, only 3 required more than 1.7 s of cpu time on the IBM 370/145. For problems with 60 objects and 60 boxes, the median time was only a little over 1 s.

- Eilon and Christofides' heuristic is of high quality. The heuristic produced only 18 suboptimal solutions. For every case in which the heuristic produced a suboptimal solution, the associated value exceeded the optimal by only 1.

- The lower bound computed by (4) is generally effective. One measure of the tightness of bounds is the ratio of the bound to the optimal value [4]. For the 165 problems, the lowest ratio ($\underline{z}_1/\bar{z}$) is 81.5% and the average ratio is 98.5%.

- The performance of the algorithm is not greatly affected by the bound. For example, in the three problems that took more than 1.7s for the algorithm, the bounds were all one less than the respective optimal values.

5. DISCUSSION

Further experiments were conducted to improve the efficiency of the algorithm. Specifically, in light of the recent bounding techniques for integer programming [4], we wanted to see if bounds better (higher) than those found by (4) could be obtained.

Following the Lagrangean relaxation approach suggested by Geoffrion [4], we constructed a Lagrangean dual formulation of (p). The obtained Lagrangean dual problem is as follows:

$$\text{(L)} \qquad \max_{\lambda} \min_{x,j} \sum_{j=1}^{n} y_j - \sum_{j=1}^{m} \lambda_i \left(\sum_{j=1}^{n} x_{ij} - 1 \right)$$

subject to

$$\text{(2)} \qquad \sum_{i=1}^{m} w_i x_{ij} \leqslant c_j y_j$$

$$\text{(3)} \qquad x_{ij} = 0 \text{ or } 1,\ y_j = 0 \text{ or } 1 \text{ for all } i,\ j.$$

Given a value for each λ_i, (L) decomposes into a collection of 0-1 knapsack problems, one for each box j. That is, for each j, we solve

$$\text{(L)} \qquad (L)_j \qquad \max \sum_{i=1}^{m} \lambda_i x_{ij}$$

subject to

$$\text{(5)} \qquad \sum_{i=1}^{m} w_i x_{ij} \leqslant c_j$$

$$x_{ij} = 0 \text{ or } 1 \text{ for all } i,\ j.$$

Let x_{ij}^* be the optimal solution and $\gamma_j = \sum_{i=1}^{m} \lambda_i x_{ij}^*$ be the optimal value of $(L)_j$. Then for each j, the optimal decisions will be

$$y_j = 1 \text{ and } x_{ij} = x_{ij}^* \text{ for } i = 1, \ldots m, \text{ if } \gamma_j > 1, \text{ and}$$

$$y_j = 0 \text{ and } x_{ij} = 0 \text{ for all } i, \text{ otherwise.} \tag{6}$$

The explanation for (6) is that if $\gamma_j > 1$, then by letting $y_j = 1$ (use box j) and assigning objects whose $x_{ij}^* = 1$, to box j, we can reduce the objective function value of (L).

To find the optimal λ (that which maximizes (L)), we use the subgradient ascent method [8]. At iteration $t \geqslant 0$, define the subgradient vector (S_i^t) to be

$$S_i^t = \sum_{j=1}^{n} x_{ij}^* - 1.$$

x_{ij}^* is the optimal solution of $(L)_j$ based on (λ_i^t). Then the next λ vector is, for each component,

$$\lambda_i^{t+1} = \lambda_i^t + \theta S_i^t, \tag{7}$$

where θ is a step size determined in the same manner described in [8]. Once λ^{t+1} is computed, we go back to solve knapsack problems $(L)_j$.

Five problems of 30 objects and 30 boxes were tested. For each of the five problems, the above procedure did not produce a bound better than that found by (4) after 300 iterations and nearly 1 cpu minute. This experience suggests that for the test problems considered in this paper, the bound derived from (4) may be the best, in terms of quality and ease of computation.

REFERENCES

[1] Brown, J. R., "Subductive Programming and Chromatic Scheduling," Ph. D. dissertation, Massachusetts Institute of Technology (1970).

[2] Eilon, S., and N. Christofides, "The Loading Problem," Management Science *17*, 259-268 (1971).

[3] Eilon, S., C. D. T. Watson-Gandy, and N. Christofides, *Distribution Management*, (Hafner, N.Y., 1971).

[4] Geoffrion, A., "Lagrangean Relaxation for Integer Programming," in *Mathematical Programming Study 2, Approaches to Integer Programming* M. L. Balinski, Ed. 82-114 (North-Holland Publishing Co., Amsterdam 1974).

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

[6] Golden, B. L., "Approaches to the Cutting Stock Problem," *AIIE Transactions, 2*, 265-274 (June 1976).

[7] Greenberg, I., "Application of the Loading Algorithm to Balance Workloads," *AIIE Transactions, 4*, 337-339 (1972).

[8] Held. M., P. Wolfe, and H. Crowder, "Validation of Subgradient Optimization" Mathematical Programming *6*, 62-88 (1974).

[9] Ingargiola, G., and J. F. Korsh, "An Algorithm for the solution of 0-1 Loading Problems," Operations Research *23*, 1110-1119 (1975).

[10] Johnson, D., "Fast Algorithms for Bin Packing," Journal of Computer and System Science, *8*, 272-314 (1974).

[11] Ross, G. T., and R. M. Soland, "A Branch and Bound Algorithm for the Generalized Assignment Problem," Mathematical Programming *8*, 91-103 (1975).