International Journal of Computer and Information Sciences, Vol. 6, No. 2, 1977

A General Packing Algorithm for Multidimensional Resource Requirements

K. Maruyama,[1] S. K. Chang,[2] and D. T. Tang[1]

Received September 1975; revised May 1976

Many problems in resource allocations, memory allocation, and distributed computer system design can be formulated as problems of packing variable-sized items into fixed-sized containers in order to minimize the total number of containers used. In this paper, a generalized packing algorithm that encompasses many well-known heuristic packing algorithms is proposed. Simulation results on this generalized packing algorithm are described, and their implications are discussed. The objective of this paper is to investigate the performance of various heuristic packing algorithms within a general framework, to obtain numerical estimates on their efficiency, and to provide guidelines on the use of these algorithms.

KEY WORDS: Algorithm; heuristic; memory allocation; packing; space utilization.

1. INTRODUCTION

Many problems in resource allocation,(1) memory allocation,(8) and distributed computer system design(2) can be formulated as problems of packing variable-sized items into fixed-sized containers in order to minimize the total number of containers used. In the literature, such problems are sometimes referred to as the box-packing problem, the stock-cutting problem,(1) the memory allocation problem,(3) or the bin-packing problem.(5) In this paper, the simple name *packing problem* will be used.

The optimal algorithm to solve the packing problem is known to be *polynomial complete*; i.e., the computation time to find the optimal packing

[1] IBM Thomas J. Watson Research Center, Yorktown Heights, New York.
[2] University of Illinois at Chicago Circle, Chicago, Illinois.

arrangement is proportional to a polynomial in n, where n is the number of items to be packed. Therefore, it is impractical to find an optimal packing arrangement when n becomes large. Moreover, when the items to be packed are not simultaneously available, the optimal algorithm is not applicable. These considerations lead us to devise various heuristic packing algorithms.

The best known heuristic packing algorithms are the best-fit (BF) algorithm, the first-fit (FF) algorithm, the next-fit (NF) algorithm, and their variants. In these algorithms, the items to be packed are arranged into an arbitrary sequence, and one item is packed at a time. In the BF algorithm, a large number of containers are given. The BF algorithm allocates an item into a container so that its remaining space is the smallest among all possible allocations. In the FF algorithm, a large number of containers are arranged into a sequence. An item is allocated to the first container in that sequence having enough space for it. In the NF algorithm, only one container is considered at a time. When no more items can be packed into it, this container is "shipped away" and another empty container is considered. The NF algorithm processes both the items and the containers sequentially. Therefore, it can be called a sequential algorithm.(2)

The items to be packed need not form an arbitrary sequence. They can be *preprocessed* into an ordered sequence based upon certain measure on the items. For example, in the one-dimensional packing problem, items can be ordered into a decreasing (nonincreasing) sequence according to their size. Improved heursitic packing algorithms can often be obtained with such preprocessing. For example, the best-fit-decreasing (BFD) algorithm is a variant of the BF algorithm with the decreasing ordering described above. Another example is the first-fit-decreasing (FFD) algorithm, which is a variant of the FF algorithm.

In general, a heuristic packing algorithm is completely characterized by its preprocessing rule; number of items considered at a time, number of containers considered at a time, and the packing rule itself. In this paper,[3] we consider the packing problem for finitely or infinitely many items, and propose a generalized heuristic algorithm that is also applicable to multi-dimensional packing problems. The generalized algorithm encompasses the FF, FFD, BF, BFD, and NF algorithms as special cases. The simulation results on the behavior of the generalized packing algorithm are described, and their implications are discussed. The objective of this paper is to investigate the performance of various heuristic packing algorithms within a general framework, to obtain numerical estimates on their efficiency, and to provide guidelines on the use of these algorithms.

[3] This paper is essentially the same as ref. 7, except that the reference includes appendices giving extensive numerical examples.

2. DESCRIPTION OF THE MODEL

2.1. A Generalized Packing Algorithm

An *item* $\bar{X}_i = (x_{i1}, x_{i2}, \ldots, x_{ij}, \ldots, x_{id})$ is represented by a d-dimensional vector, where x_{ij}, $0 \leqslant x_{ij}$, $\leqslant 1$, denotes the jth requirement of the ith item, and $\sum_{j=1}^{d} x_{ij} > 0$. The *capacity* of a *container*, $\bar{C}_k = (C_{k1}, C_{k2}, \ldots, C_{kj}, \ldots, C_{kd})$, is also a d-dimensional vector, where C_{kj} denotes the *current capacity* of the jth component of the kth container, $0 \leqslant C_{kj} \leqslant 1$. The original capacity of a free *container* (or an empty container) is a vector $\bar{C}_k = (1, 1, \ldots, 1, \ldots, 1)$. An item $\bar{X}_i$ can be *packed* (or fit) into a container $\bar{C}_k$, if $\bar{C}_k \geqslant \bar{X}_i$, or $C_{kj} \geqslant X_{ij}$ for all j. The items are obtained from a source G, and the total number of items to be packed is denoted by n.

A schematic diagram of the *generalized packing algorithm* is shown in Fig. 1. The items obtained from the source G are first stored in an *item list* of length N. These items are then preprocessed according to a specified *preprocessing rule* W, which is described in Sec. 2.2. After preprocessing, an item in the item list is selected and packed into one of the M containers that can accomodate it. The selection and packing mechanism is speified by a *packing rule* P which is described in the next paragraph. If no container can accomodate this item, then it stays in the item list, and another item is selected and packed using the packing rule. Whenever an item has been successfully packed, it is checked whether R of the N items in the item list has been packed into the containers. If this is the case, then another R new items are obtained from the source G and stored in the item list. The new

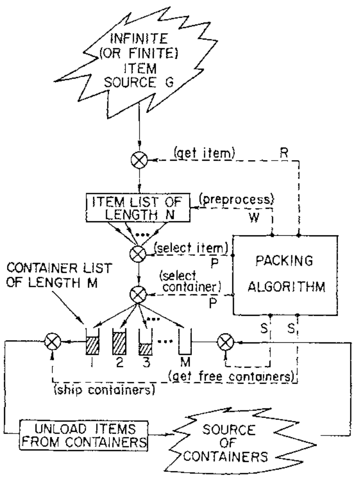

Fig. 1. Schematic diagram of the generalized packing algorithm.

item list is again preprocessed according to the preprocessing rule W. When no more items can be packed into the containers, the first S containers of the M containers are "shipped away," and another S free containers are added at the end of the remaining $M - S$ containers to form a new container list. The algorithm repeats the above described operations until all of the n items have been packed into the containers. When this happens, all the non-empty containers in the container list are shipped away, and the algorithm terminates. The total number of containers used is defined to be the total number of containers being shipped.

As explained above, how to select an item from the item list and pack it into one of the M containers is specified by the *packing rule P*. In Fig. 2, the flowchart for the *global packing rule* P_G is shown. This packing rule is a generalization of the best-fit algorithm, and the packing decision is made by examining all M containers in the container list. In Fig. 2, a function v is used to denote a scalar measure of item requirements, and this is discussed in the following section. The ith item $\bar{X}_i$ is packed into the kth container $\bar{C}_k$ if $v(\bar{C}_k - \bar{X}_i)$ is the smallest among all $v(\bar{C}_j - \bar{X}_i)$ of M containers. In Fig. 3, the flowchart for the *local packing rule* P_L, which is essentially the first-fit algorithm, is shown. In this rule, the packing decision is made by examining the containers in a fixed sequence, and the ith item is packed into the first container having enough space for it. In both packing rules, the items are examined one by one, in the same sequence as they appear in the item list.

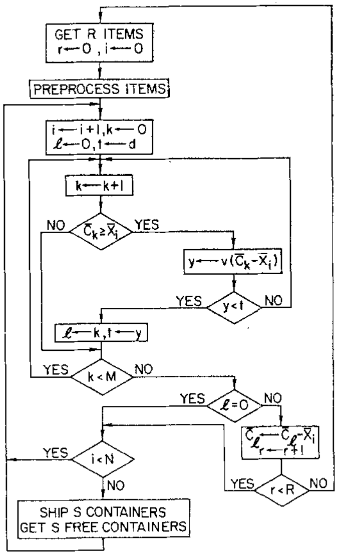

Fig. 2. Global packing rule.

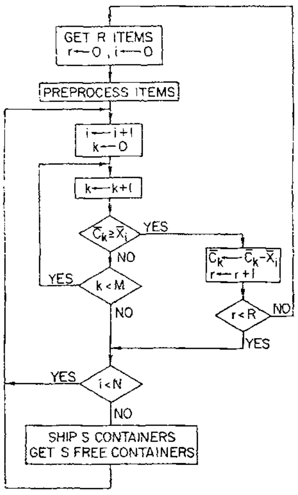

Fig. 3. Local packing rule.

The generalized packing algorithm, therefore, can be parametrically specified by $A(N, R, M, S, W, P)$,

N	is the number of items in the item list,
R	is the number of items to be packed before the item list is refilled, $1 \leqslant R \leqslant N$,
M	is the number of containers in the container list,
S	is the number of containers to be shipped when no more items from the item list can be packed into the containers, $1 \leqslant S \leqslant M$,
W	is the preprocessing rule,
P	is the packing rule, P_G being the global packing rule (Fig. 2) and P_L being the local packing rule (Fig. 3).

2.2. The Preprocessing Rules

An intuitively appealing preprocessing operation is to order the items according to a certain measure on the items. For example, when the items are scalars ($d = 1$), the items can be ordered into a nonincreasing sequence according to the magnitude of the X_i.[3-5] All preprocessing rules discussed in this section are based upon this ordering operation.

When the items are multidimensional vectors, various scalar measures v_l on $\bar{X}_i$ yield different preprocessing rules. The $\bar{X}_i$ are sorted according to $v_l(\bar{X}_i)$ into a nonincreasing sequence. The preprocessing rules used in the simulation experiments are summarized in Table I, where the *preprocessing*

Table I. Preprocessing Rules

W	Measure $v_i(\bar{X}_i)$	Explanation	Preprocessing cost
W_0	$v_0(\bar{X}_i) = N - i$	Items ordered according to actual arrival order, i.e., no preprocessing	$c_0 d + \log_2 N$
W_1	$v_1(\bar{X}_i) = \sum_{j=1}^{d} x_{ij}$	Items ordered according to sum of x_{ij}	$c_1 d + \log_2 N$
W_2	$v_2(\bar{X}_i) = \sum_{j=1}^{d} (x_{ij})^2$	Items ordered according to sum of squares of x_{ij}	$c_2 d + \log_2 N$
W_3	$v_3(\bar{X}_i) = \sum_{j=1}^{d} (x_{ij})^3$	Items ordered according to sum of cubes of x_{ij}	$c_3 d + \log_2 N$
W_4	$v_4(\bar{X}_i) = \sum_{j=1}^{d} (x_{ij})^4$	Items ordered according to sum of fourth powers of x_{ij}	$c_4 d + \log_2 N$
W_5	$v_5(\bar{X}_i) = \max_j \{x_{ij}\}$	Items ordered according to maximum of x_{ij}	$c_5 d + \log_2 N$
W_6	$v_6(\bar{X}_i) = x + \alpha\, \sigma$ $0 \leqslant \alpha \leqslant 1$	Items ordered according to average x of x_{ij} plus its standard deviation σ	$c_6 d + \log_2 N$
W_7	$v_7(\bar{X}_i) = \prod_{j=1}^{d} x_{ij}$	Items ordered according to product of x_{ij}	$c_7 d + \log_2 N$
W_8	$v_8(\bar{X}_i) = \sum_{j=1}^{d} w_j x_{ij}$	Items ordered according to weighted sum $w_j x_{ij}$, $w_j \geqslant 0$	$c_8 d + \log_2 N$

time refers to the computation time needed to perform the preprocessing operations per item.

2.3. Discussions on the Generalized Packing Algorithm

From Secs. 2.1 and 2.2, it can be seen that the generalized packing algorithm A becomes equivalent to other heuristic algorithms when its parameters are specified accordingly. The following are the most important cases for one-dimensional packing problems, i.e., for $d = 1$:

1. $A(1, 1, 1, 1, W_0, P_L)$ and $A(1, 1, 1, 1, W_0, P_G)$ are the next-fit algorithms.
2. $A(1, 1, k, 1, W_0, P_L)$ is the next-k-fit algorithm.(5)
3. $A(n, n, 1, 1, W_0, P_L)$, $A(1, 1, n, n, W_0, P_L)$, and $A(n, n, 1, 1, W_0, P_G)$ are the first-fit algorithms.
4. $A(n, n, 1, 1, W_1, P_L)$ and $A(n, n, 1, 1, W_1, P_G)$ are the first-fit-decreasing algorithms.
5. $A(1, 1, n, n, W_0, P_G)$ is the best-fit algorithm.
6. $A(n, n, n, n, W_1, P_G)$ is the best-fit-decreasing algorithm.
7. $A(N, R, 1, 1, W, P_L)$ and $A(N, R, 1, 1, W, P_G)$ are equivalent.

2.4. Packing Efficiency

Since it is assumed that the total number of items n to be packed is either unknown or infinite, it is impossible to determine the minimum number of containers needed to pack n items for the optimum packing. Therefore, the number of "excess" containers needed to pack n items cannot be used as a measure of packing efficiency. Instead, we use *space utilization* to measure the packing efficiency of a given packing algorithm.

The space utilization U_A is defined as

$$U_A = \frac{\max_j(\sum_{i=1}^{n} x_{ij})}{\begin{array}{l}\text{number of containers required to pack } n \text{ items} \\ \text{using algorithm } A(N, R, M, S, W, P)\end{array}}$$

where $0 < x_{ij} \leqslant 1$ for all i and j. Clearly, the space utilization U_A ranges between 0 and 1.[4] For example, all of the next-fit, next-k-fit, first-fit, first-fit-decreasing, best-fit, and best-fit-decreasing algorithms guarantee that the space utilization U_A lies between 0.5 and 1.0 when $d = 1$ (one-dimensional case).(2–5,7) It should be noted that a small U_A does not necessarily imply that

[4] It can be easily shown that the next-fit algorithm guarantees the space utilization $U_A \geqslant 1/(2d)$.

the algorithm is bad. In fact, the space utilization U_{op} of an optimum packing algorithm, in some cases, can be very close to 0.5.

Another measure for the efficiency of a given heuristic packing algorithm is the estimated *average computation time* required to pack an item. The computation time generally consists of two parts: the time associated with preprocessing and the time associated with packing. The average preprocessing time per item, as shown in Table I, has a form of $c_1 d + \log_2 N$, where c_1 is a constant, d is the dimension of items, and N is the length of the item list. The average packing time per item is approximately $c_2 dMN$, where c_2 is a constant and M is the length of the container list. Therefore, the average computation time per item with preprocessing is approximately $c_1 d + \log_2 N + c_2 dMN$. In practice, it is required that a packing algorithm achieve certain packing rate, i.e., at least m items per unit time. Thus, the parameters N and M must at least satisfy $c_1 d + \log_2 N + c_2 dMN \leqslant 1/m$. Furthermore, M and N should be chosen so as to maximize the space utilization U_A.

In the following section, the effects of parameters N, R, M, S, W, and P on the space utilization U_A are observed via simulation experiments. It is hoped that these results will be useful in selecting appropriate algorithms in different applications.

3. EXPERIMENTAL RESULTS

The generalized packing algorithm described in Sec. 2 was coded in *PL/*1, and simulation experiments were run on an IBM 370/168 computer. In the simulation, requirements are generated by a source that generates numbers having various statistical distributions. For d-dimensional items, each requirement x_{ij}, of an item $\bar{X}_i$ is generated independently by a source G_j, for $1 \leqslant j \leqslant d$.

The source G_j is specified by four parameters (k, m, a, b). To produce a number x_{ij} when k is 1, the source G_j takes the average of m random numbers that are uniformly distributed between 0 and 1, multiplies the resulting average by a, and adds b to the product. In other words, if the m uniformly distributed random numbers are $y_1, \ldots, y_2, \ldots, y_m$, then

$$x_{ij} = a\left(\frac{\sum_{i=1}^{m} y_i}{m}\right) + b \qquad [G_j(1, m, a, b)]$$

Similarly, to produce a number x_{ij} when k is 2, the source G_j takes the *product* of m uniformly distributed random numbers, multiplies the resulting product by a and adds b to the new product. In other words,

$$x_{ij} = a\left(\prod_{i=1}^{n} y_i\right) + b \qquad [G_j(2, m, a, b)]$$

In our experiments, we take $b < a$ and $0 < a + b \leqslant 1$. The number m is typically between 1 and 10. The source $G_j(1, m, a, b)$ produces numbers with a bell-shaped distribution between b and $a + b$, and $G_j(2, m, a, b)$ produces numbers with a skewed distribution having a long tail between b and $a + b$. These empirical distributions are thought to be of practical significance.

As explained in Sec. 2.1, the algorithm $A(N, R, M, S, W, P)$ is controlled by the parameters N, R, M, S, W, and P. In the simulation, these parameters and the sources $G_j(k, m, a, b)$ are varied so that the packing efficiency of many heuristic packing algorithms can be observed. In what follows, the experimental results are illustrated by several figures. The detailed experimental data can be found in ref. 7.

We first describe simulation results for the one-dimensional case ($d = 1$). The simplest packing algorithm is $A(1, 1, 1, 1, W_0, P_L)$, or the next-fit algorithm. For a uniform distribution for the items ($G_1(1, 1, a, b)$), Fig. 4 illustrates the space utilization U_A for various values of a and b. The items are uniformly distributed between b and $a + b$. For a fixed b, the *worst-case space utilization* is the smallest U_A achieved by changing a. As b increases, it can be seen that the worst-case space utilization decreases. Figure 4 shows that packing efficiency generally decreases when the average requirement increases. When the requirements are small (say $b \leqslant 0.1$ and $a \leqslant 0.1$), high packing efficiency ($U_A > 0.95$) can be achieved.

The next algorithm investigated is the first-fit algorithm, or $A(n, n, 1, 1, W_1, P_L)$. Figure 5 illustrates the space utilization for various values of n (total number of items packed) and three different sources, $G_1(1, 1, 0.25, 0.25)$, $G_1(1, 1, 1, 0)$, and $G_1(1, 1, 0.25, 0)$. The space utilization generally approaches a certain value as n grows large, say 1000. For some distributions, it can be seen that the space utilization is quite insensitive to n.

The algorithm $A(n, n, 1, 1, W_1, P_L)$ can be refined by changing it to $A(N, N, 1, 1, W_1, P_L)$. In other words, instead of preprocessing all the n

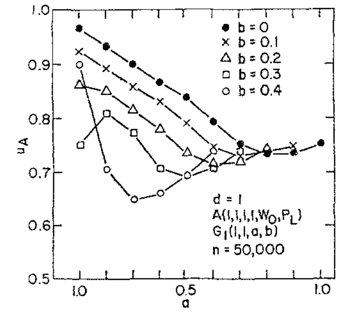

Fig. 4. The effect of a and b.

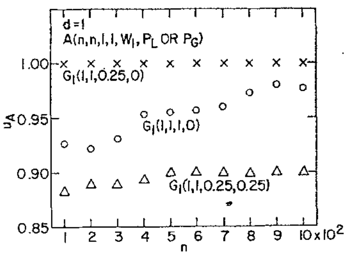

Fig. 5. The effect of the number of items to be packed.

items, only N items will be preprocessed at a time. By doing so, a saving in preprocessing time can be achieved. Also, a limited buffer can be used to store the item list (see Fig. 1). Figure 6 illustrates the effect of N on the space utilization. As N increases, the space utilization increases. For $N = 1$, U_A is approximately 0.75, and for $N = n$, U_A is approximately 0.98. To achieve a utilization, $(0.98 + 0.75)/2$ or 0.86, $N = 6$ is sufficient. In other words, a small item list of size N can yield a significant improvement in packing efficiency.

For the multidimensional case ($d \geqslant 1$), the behavior of the algorithms $A(N, N, 1, 1, W_0, P_L)$ and $A(N, N, 1, 1, W_1, P_L)$ is again observed, assuming different sources G_j. The simulation results are shown in Figs. 7 and 8. When a small item list of length no larger than 100 ($N \leqslant 100$) is used, space utilization can be improved significantly by preprocessing items. Even with no preprocessing of items, an improvement in space utilization is still possible for small increases of N ($N \leqslant 50$). In general, an increase in the dimensionality d will reduce the space utilization. The amount of reduction, however, depends on the distribution of these requirements. Sometimes the effect may be negligible. For example, the space utilization for $d = 2$ of Fig. 8 is approximately the same as the space utilization for $d = 1$ of Fig. 7.

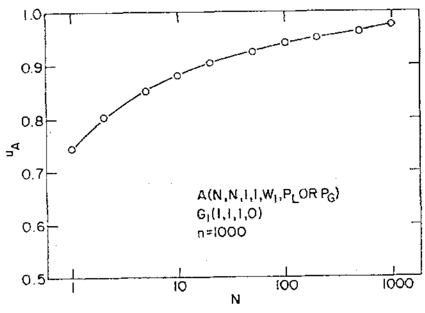

Fig. 6. The effect of N for small n.

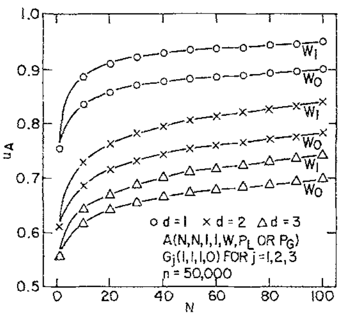

Fig. 7. The effect of *N*.

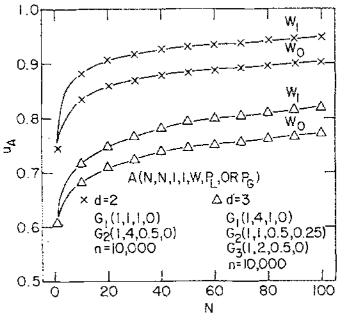

Fig. 8. The effect of *N*.

Similarly, the space utilization for $d = 3$ of Fig. 8 is approximately the same as the space utilization for $d = 2$ of Fig. 7. This can be explained as follows. For the case of $d = 2$ in Fig. 8, $G_1(1, 1, 1, 0)$ and $G_2(1, 4, 0.5, 0)$ are used, where G_1 has mean 0.5 and G_2 has mean 0.25. When an item (x_{i1}, x_{i2}) is packed into a container (C_{k1}, C_{k2}), the first requirement x_{i1} has a much higher probability to exceed the current capacity $(C_{k1} < x_{i1})$ than the second requirement. One might say that $G_1(1, 1, 1, 0)$ is "dominant" over $G_2(1, 4, 0.5, 0)$. Thus, even though the dimension is 2, the space utilization given for this case is very close to $d = 1$ with $G_1(1, 1, 1, 0)$ of Fig. 7.

Let us observe the effect of R on the space utilization, using the algorithm $A(N, R, 1, 1, W_1, P_L)$. The simulation results for dimensions $d = 2$ and 3 and for various item distributions are shown in Fig. 9. It can be observed that significant improvements on U_A can be achieved by decreasing R from N to 1, i.e., by increasing the frequency of refilling the item list. The improvement is more significant for smaller N. For larger N, the significant improvements were achieved when $N - 5 \leqslant R \leqslant N$. This means that one should refill the item list when about five items are left. It can be observed

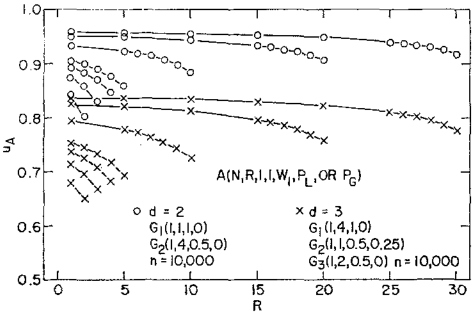

Fig. 9. The effect of R.

also that the improvement achieved by decreasing R is greater than the improvement achieved by increasing N.

The effect of M and N on the space utilization is shown in Fig. 10 using the algorithm $A(N, N, M, M, W, P_L)$. It can be observed that by increasing M from 1 to 8, U_A can be improved up to 10%. However, the improvement decreases as N increases. This shows that for relatively large N, U_A is not sensitve to the change of M. On the other hand, for large M, U_A is not sensitive to the change of N. Thus, it is desirable to increase either N or M, but not both, to attain higher space utilization. Let us further observe the effect of M on the space utilization as shown in Fig. 11, which only the space utilization given by the algorithm $A(1, 1, M, M, W_0, P_L)$ in which uses the local packing rule are shown. However, it has been observed from our simulation experiments that the algorithm $A(1, 1, M, M, W_0, P_G)$ which uses the global packing rule gives up to 0.6% better space utilization than the one given by $A(1, 1, M, M, W_0, P_L)$. From Fig. 11, it can be observed that the amount of improvement on U_A that can be achieved by increasing M depends on the requirement distributions and the dimension d. In general,

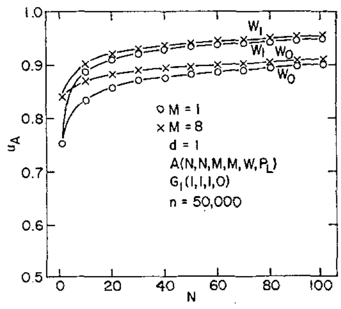

Fig. 10. The effect of N and M.

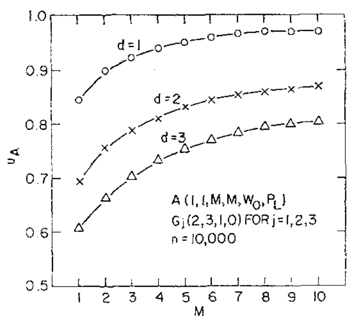

Fig. 11. The effect of M.

less improvement on the space utilization can be achieved by increasing N than by increasing M.

Let us observe the effect of S on the space utilization. In Fig. 12, space utilization using the algorithm $A(N, N, 10, S, W_1, P_L)$ for $1 \leqslant S \leqslant M = 10$ is shown. As S decreases from M to 1, U_A increases. For $N \geqslant 10$, the effect of S on U_A is very little. Even for $N < 10$, the effect of S on U_A is not as large as the effect of M on U_A.

To determine the simplest and yet the most suitable preprocessing rules, many simulation experiments were run using the algorithm $A(N, N, 1, 1, W, P_L)$ for those preprocessing rules W of Table I. It has been observed that for dimension d, $d \geqslant 2$, and for various requirement distributions, the preprocessing rules W_1 through W_6 are far better than the preprocessing rule W_7. It has been observed also that which of $W_1, W_2, \ldots, W_6$ gives the best space utilization depends on the requirement distribution. However, there was no more than 0.5 % difference between the space utilization given by these preprocessing rules. In general, the simple preprocessing rules W_1 and W_2 are good choices.

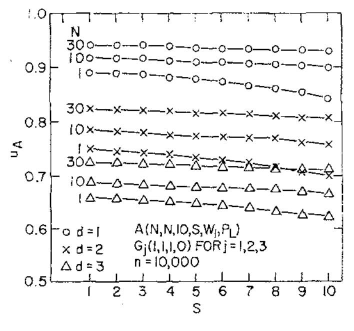

Fig. 12. The effect of S.

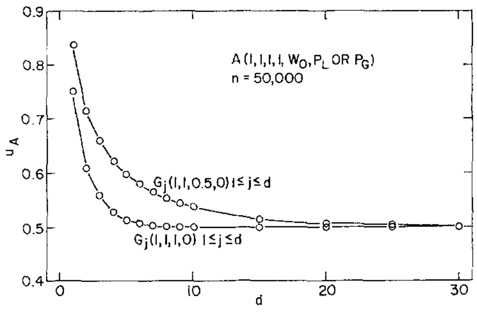

Fig. 13. The effect of dimension d.

Finally, let us observe the effect of the dimension d on the space utilization. Figure 13 shows U_A using the next-fit algorithm, or $A(1, 1, 1, 1, W_0, P_4)$ for $d \geqslant 1$. As d increases, the space utilization U_A decreases rapidly and approaches to 50%. In general, it is quite difficult to find an example for small d such that the space utilization given by the next-fit algorithm is less than 50%. One example that gave a space utilization slightly below 50% is the case with $d = 10$ and skewed distributions, $G_j(2, m, 1, 0)$ for $m = 2$.

4. DOMINANCY AND PACKING EFFICIENCY

In Sec. 3, it was observed that for multidimensional packing problems, packing efficiency seems to be determined by a few *dominant* requirements in the requirement vector. The *degree of dominancy* p_j of the jth requirement is the conditional probability that the jth requirement will exceed the capacity, given that the requirement vector $\bar{X}_i$ cannot fit in the capacity vector of container $\bar{C}_k$. For a given item sequence $\bar{X}_1, \bar{X}_2, \ldots, \bar{X}_n$, let $\{\bar{X}_i, i_k \leqslant i \leqslant i_{k+1} - 1\}$ be a subsequence of items packed into the kth container for $1 \leqslant k \leqslant m$, where $i_1 = 1$ and $i_{m+1} - 1 = n$. Then we compute the degree of dominancy of the jth requirement p_j, $1 \leqslant j \leqslant d$, in the following manner:

$$p_j = \frac{\sum_{k=1}^{m-1} \lceil \sum_{i=1_k}^{i_{k+1}} x_{ij} - 1 \rceil}{m - 1}$$

where $\lceil \ \rceil$ denotes the ceiling function. Thus,

$$\left\lceil \sum_{i=i_k}^{i_{k+1}} x_{ij} - 1 \right\rceil = 1$$

if $\sum_{i=i_k}^{i_{k+1}} x_{ij} > 1$, and 0 otherwise. In other words, the sum $\sum_{k=1}^{m} \lceil \sum_{i=i_k}^{i_{k+1}} x_{ij} - 1 \rceil$ counts the number of times a container is filled due to excess in the jth requirement. Then p_j gives a normalized value for $\sum_{k=1}^{m} \lceil \sum_{i=i_k}^{i_{k+1}} x_{ij} - 1 \rceil$.

Clearly, the larger p_j is, the more containers are filled due to excess in the jth requirement, and the more dominant the jth requirement is. If $p_j > p_k$, then the jth requirement is said to *dominate* the kth requirement.

The degree of dominancy can be measured by running a packing algorithm to pack n items. For example, the next-fit algorithm $A(1, 1, 1, 1, W_0, P_L)$ is run to pack items with three requirements. In this experiment, with $d = 3$, $G_1(1, 1, 1, 0)$, $G_2(1, 2, 0.7, 0)$, $G_3(2, 3, 0.9, 0.1)$, and $n = 10{,}000$, the degrees of dominancy are $p_1 = 0.889$, $p_2 = 0.322$, and $p_3 = 0.049$. Here G_1 is a uniform distribution on $[0, 1]$, G_2 is a bell-shaped distribution on $[0, 0.7]$, and G_3 is a skewed distribution on $[0.1, 0.9]$. The first requirement dominates the second requirement, and the second dominates the third.

Let us take advantage of this dominancy information to determine the weight w_j for the preprocessing rule W_8 with the measure $\sum_{j=1}^{d} w_j x_{ij}$, to achieve better space utilization U_A. One approach is to set $w_j = 1$ for one or two dominant requirements ($w_j = 0$ for others). This has the advantage in its simplicity but suffers from potential inferior performance, especially when d is large and dominancy figures are not widely separated. Another approach, which is perhaps more natural than the above 0 or 1 selection on the weight w_j, is to set $w_j = p_j$, i.e., to use the preprocessing measure $\sum_{j=1}^{d} p_j x_{ij}$. Let us observe the effect of these preprocessing measures on space utilization. In Table II, the space utilization for the algorithm $A(N, N, 1, 1, W_8, P_L)$ on items generated from sources $G_1(1, 1, 1, 0)$, $G_2(1, 2, 0.7, 0)$, and $G_3(2, 3, 0.9, 0.1)$ is shown. In each column except the last, a different subset of requirements is considered for the preprocessing rule W_8; i.e., items are sorted into nonincreasing order by $\sum_{j \in J} x_{ij}$, where J is a subset of $\{1, 2, 3.\}$. In the last column, the preprocessing rule W_8 with the measure $\sum_{j=1}^{d} p_j x_{ij}$, is considered. Further simulation results based on this measure can be found in ref. 7. From these results, one can observe the following:

1. Preprocessing requirements with larger degrees of dominancy give better packing efficiency than preprocessing requirements with smaller degrees of dominancy. In Table II, preprocessing that includes at least the first requirement (which is the most dominant requirement) gives better packing efficiency than all other cases.

2. Preprocessing more requirements does not necessarily give better packing efficiency than preprocessing some subset of more dominant requirements. In Table II, the packing efficiency given by preprocessing for the first requirement (second column) is higher than the one given by preprocessing all the requirements (eighth column).

3. Preprocessing less dominant requirements gives worse packing efficiency than no preprocessing (e.g., in Table II, for $N \geqslant 9$, the first column shows better packing efficiency than the third column).

Table II. Space Utilization by $A(N, N, 1, 1, W_8, P_L)$ with Different Combination of Requirements for Preprocessing[a]

N	Different combination of requirements for preprocessing $\bar{w} = (w_1w_2w_3)$ 000	100	010	001	110	101	011	111	$p_1p_2p_3$
1	0.712	0.712	0.712	0.712	0.712	0.712	0.712	0.712	0.712
2	0.738	0.767	0.742	0.737	0.765	0.764	0.741	0.763	0.767
3	0.754	0.796	0.761	0.756	0.794	0.792	0.761	0.791	0.797
4	0.768	0.815	0.775	0.768	0.812	0.809	0.773	0.808	0.816
5	0.778	0.826	0.782	0.776	0.824	0.821	0.781	0.819	0.829
6	0.787	0.836	0.792	0.786	0.835	0.831	0.790	0.829	0.840
7	0.793	0.844	0.796	0.793	0.841	0.838	0.795	0.837	0.848
8	0.799	0.849	0.802	0.798	0.850	0.844	0.802	0.843	0.855
9	0.808	0.854	0.803	0.803	0.853	0.848	0.806	0.848	0.858
10	0.808	0.860	0.808	0.804	0.856	0.852	0.807	0.853	0.862
15	0.826	0.873	0.815	0.820	0.875	0.869	0.818	0.869	0.880
20	0.835	0.887	0.822	0.830	0.887	0.883	0.827	0.883	0.895
30	0.851	0.902	0.832	0.843	0.903	0.899	0.838	0.895	0.910
40	0.862	0.912	0.838	0.851	0.913	0.907	0.843	0.905	0.920
50	0.869	0.919	0.844	0.857	0.919	0.914	0.850	0.914	0.927
100	0.891	0.937	0.854	0.878	0.936	0.932	0.868	0.930	0.945

[a] $d = 3$, $G_1(1, 1, 1, 0)$, $G_2(1, 2, 0.7, 0)$, $G_3(2, 3, 0.9, 0.1)$, $n = 10{,}000$, $p_1 = 0.889$, $p_2 = 0.322$, $p_3 = 0.049$.

4. The measure $\sum_{j=1}^{d} p_j x_{ij}$ gives better packing efficiency than the measure $\sum_{j=1}^{d} w_j x_{ij}$ for $w_j = 0$ or 1 in all the cases observed. This approach is also instinctively appealing since the measure is symmetrical with respect to j and is therefore universally applicable regardless of how dominancy figures vary.

5. SUMMARY AND DISCUSSION

We have proposed a generalized heuristic packing algorithm $A(N, R, M, S, W, P)$ with multiple constraints which encompasses as special cases many well-known heuristic algorithms including the next-fit algorithm, the first-fit algorithm, the first-fit-decreasing algorithm, the best-fit algorithm, and the best-fit-decreasing algorithm. The average computation time per item by the algorithm is approximately $c_1 d + \log_2 N + c_2 dNM$. Thus, the computation time can be controlled by restricting parameters N and M. To measure packing efficiency, the space utilization U_A was computed for various parameter values and preprocessing rules as well as for different number of constraints and different source distributions.

The following observations summarize our study on the behavior of various heuristic algorithms evaluated via simulation experiments. It is hoped that these observations can be used to determine the appropriate values of the parameters N, R, M, S, W, and P, in the selection of the best algorithms in different applications.

1. The simplest next-fit algorithm, $A(1, 1, 1, 1, W_0, P_L)$, gives better than 50% space utilization when $d = 1$.[2] Experimentally, it also gives better than 50% space utilization, even for multidimensional cases ($d \geqslant 2$).
2. By preprocessing items, we can expect a significant improvement on space utilization. Since the computation time associated with simple preprocessing operations is small, it is desirable to preprocess items using a small item list with $N \leqslant 100$.
3. The space utilization given by the algorithm $A(N, R, M, S, W, P)$ for $W = W_0$, W_1, and $P = P_L$ (or $P = P_G$) ranges between that of the next-fit algorithm and that of the first-fit-decreasing algorithm (or the best-fit-decreasing algorithm). When the number of items to be packed is large or unknown, an effective approach to improve the space utilization U_A is to decrease R. Another approach is to increase N, and still others are to increase M and to decrease S. One may thus adjust these parameters in that sequence to achieve desired performance.
4. In practice, parameters such as N, M, W, and P of the algorithm $A(N, R, M, S, W, P)$ are limited by available work space and allowable computation time. In such cases, the space utilization can be improved by first decreasing the parameter R and then decreasing the parameter S.

In the simulation experiments described above, packing efficiency was measured by the space utilization U_A. Several theoretical studies[2–5,7] on packing algorithms have used the *packing ratio* r_A as a measure of packing efficiency. Here r_A is defined to be the ratio of the number of containers required to pack n items using algorithm A, over the number of containers needed by the *optimal packing algorithm*. From Sec. 2.3, it can be seen that

$$r_A = \frac{1}{U_A} \, \frac{\max_j \left(\sum_{i=1}^{n} x_{ij}\right)}{\begin{array}{c}\text{number of containers needed by}\\ \text{the optimal packing algorithm}\end{array}}$$

Since $\max_j(\sum_{i=1}^{n} x_{ij})$ is the sum of the requirements for the dimension that needs the largest capacity, the number of containers needed by the optimal packing algorithm cannot be less than $\max_j(\sum_{i=1}^{n} x_{ij})$. Therefore,

$$r_A \leqslant \frac{1}{U_A}$$

This inequality shows that the number $1/U_A$ gives an upper bound to the packing ratio r_A. The space utilization U_A, therefore, can be used to relate empirical results to theoretical analyses. For the one-dimensional case, Ullman[8] shows that the best-fit and first-fit algorithms have $r_A \leqslant 1.7$. Johnson[4,5] shows that both best-fit-decreasing and first-fit-decreasing algorithms have $r_A \leqslant 11/9$. Chang and Tang[2] and Johnson[4,5] have both shown that the next-fit algorithm has $r_A \leqslant 2$. For the multidimensional case, Yao[9] recently showed that the first-fit algorithm has $r_A \leqslant d + 1$ for large n, where d is the dimension of items. Kou and Markowsky[6] give a similar result for a merge algorithm[2] that merges items in different containers as long as possible.

Comparing these theoretical results to our simulation results, we can see that the heuristic algorithms usually behave far better than the theoretical bounds, especially when the item requirements have independent, uniform (or symmetric) distribution. In the multidimensional case, the bound $d + 1$ is clearly seldom reached. In other words, the space utilization does not decrease as rapidly as the dimensionality. In practice, one or two requirements of the items will become dominant, and the packing efficiency in terms of U_A or r_A really depends upon only the dominant item requirements. This suggests that, in practice, one may first look at the item requirements to pick up the dominant ones. A heuristic packing algorithm tuned to these requirements can then be selected.

A general conclusion of this study on packing algorithms is that, in practice, it does not seem necessary to use very sophisticated packing algorithms to achieve a reasonable packing efficiency. The algorithm $A(N, R, N, S, W_8, P_L)$, i.e., a generalized next-fit-decreasing packing algorithm with the measure $v_8(\bar{X}_i) = \sum_{j=1}^{d} p_j x_{ij}$, appears to be a generally efficient algorithm for most applications. The parameters N, R, M, and S may be selected according to our observations described earlier in this section. It has been shown earlier that one can implement a generalized algorithm$A$ that is able to switch from the next-fit mode to the first-fit mode or the best-fit mode. Thus, when the total number of items to be packed is infinitely large, the algorithm A first packs some items in the next-fit mode and also measures the degrees of dominancy p_j, $j = 1, 2,..., d$. Then the algorithm A adapts the preprocessing measure $v_8(\bar{X}_i)$ and packs the rest of items by changing its packing mode. When the distribution of each requirement changes periodically, the algorithm A can be implemented in such a way to observe the degrees of dominancy periodically to update the scalar measure in order to maintain high packing efficiency.

REFERENCES

1. A. R. Brown, *Optimum Packing and Depletion* (MacDonald, London, and American Elsevier, New York, 1971).

2. S. K. Chang and D. T. Tang, "Processor Allocation in a Distributed Computer System," *Proceedings of the IEEE/NBS Symposium on Computer Communications: Trends and Applications* (1975).
3. M. R. Garey, R. L. Graham, and J. D. Ullman, "Worst-case analysis of memory allocation algorithms," *Proceedings of the Fourth Annual ACM Symposium on the Theory of Computing* (1972).
4. D. S. Johnson, "Fast allocation algorithms," *Thirteenth Annual IEEE Symposium on Switching and Automata Theory* (1973), pp. 144–154.
5. D. S. Johnson, "Near-Optimal Bin Packing Algorithms," Ph.D. Thesis, MIT, Cambridge, Mass. (June 1973).
6. L. T. Kou and G. Markowsky, "Multi-dimensional Bin Packing Algorithms," RC 5391, IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y. (June 1975).
7. K. Maruyama, S. K. Chang, and D. T. Tang, "An Experimental Study on a Generalized Packing Algorithm," RC 5412, IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y. (May 1975).
8. J. D. Ullman, "The Performance of a Memory Allocation Algorithm," Technical Report No. 100, Princeton University, Princeton, N.J., 1971.
9. A. Yao, "Bin Packing Problem," presented at IBM Research Center, Yorktown Heights, N.Y. (November 1974).