FAST APPROXIMATION ALGORITHMS FOR KNAPSACK PROBLEMS

by

E. L. Lawler

Memorandum No. UCB/ERL M77/45

21 June 1977

FAST APPROXIMATION ALGORITHMS FOR KNAPSACK PROBLEMS

by

E. L. Lawler

Memorandum No. UCB/ERL M77/45

21 June 1977

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

FAST APPROXIMATION ALGORITHMS FOR KNAPSACK PROBLEMS*

E.L. Lawler

Computer Science Division
and the Electronics Research Laboratory
University of California at Berkeley

ABSTRACT

Fully polynomial approximation algorithms for knapsack problems are presented. These algorithms are based on ideas of Ibarra and Kim, with modifications which yield better time and space bounds, and also tend to improve the practicality of the procedures. Among the principal improvements are the introduction of a more efficient method of scaling and the use of a median-finding routine to eliminate sorting. The 0-1 knapsack problem, for n items and accuracy $\varepsilon > 0$, is solved in $(n \log(\frac{1}{\varepsilon}) + \frac{1}{\varepsilon^4})$ time and $O(n + \frac{1}{\varepsilon^3})$ space. The time bound is reduced to $O(n + \frac{1}{\varepsilon^3})$ for the "unbounded" knapsack problem. For the "subset-sum" problem, $O(n + \frac{1}{\varepsilon^3})$ time and $O(n + \frac{1}{\varepsilon^2})$ space, or $O(n + \frac{1}{\varepsilon^2} \log(\frac{1}{\varepsilon}))$ time and space, are achieved. The "multiple choice" problem, with m equivalence classes, is solved in $O(\frac{nm^2}{\varepsilon})$ time and space.

*The research reported in this paper was supported in part by National Science Foundation Grant MCS76-17605, and in part by the Mathematical Center, Amsterdam, The Netherlands, where the author was a visitor, January-February, 1977.

1. Introduction

The "0-1" knapsack problem is as follows: Given n pairs of positive integers, (p_j,a_j) and a positive integer b, find $x_1,x_2,\ldots,x_n$ so as to

$$\begin{aligned} &\text{maximize} && P = \textstyle\sum_j p_j x_j \\ &\text{subject to} && A = \textstyle\sum_j a_j x_j \le b \ , \end{aligned} \qquad x_j \in \{0,1\} \ .$$

We may think of j as indexing *items*, with associated *profits* p_j and *weights* a_j. The object is to find the most profitable possible selection of items which can be made to fit into a knapsack with capacity b. One variation of the problem permits items to be chosen with repetition. That is, x_j is permitted to be any nonnegative integer. This is sometimes called the "unbounded" knapsack problem.

There are well-known methods for solving the 0-1 knapsack problem which have worst-case running time of $O(nb)$ [2]. However, these do not qualify as "polynomial-bounded" algorithms, because the running time is not bounded by a polynomial in the length of the excoding of input data. Thus $O(n \log b)$ is a polynomial bound, but not $O(nb)$. (Unless data are encoded in unary notation, a possibility we disregard here, cf. [3].) In fact, the problem is known to be NP-complete, even in the case of the "subset-sum" problem, where $p_j = a_j$, for all items. Hence it is very unlikely that any polynomial-bounded algorithm exists.

However, it is possible, within polynomial time, to find a solution which is arbitrarily close to optimum. An approximation algorithm for this purpose receives two inputs: one is the encoded set of data for a problem instance (i.e. pairs (p_j,a_j), and b), the other is a number ε, $0 < \varepsilon \le 1$, which prescribes the degree of accuracy required. The algorithm then

produces a solution with profit P, such that if P^* is the value of an optimal solution,

$$P^* - P \leq \varepsilon P^* .$$

If for every fixed ε, the algorithm operates in time bounded by the length of the encoded input, the algorithm is a "polynomial time approximation scheme." If the algorithm operates in time bounded by a polynomial in the length of the encoded input and in $1/\varepsilon$, it is said to be "fully polynomial" [3].

Ibarra and Kim [4] have presented fully polynomial approximation algorithms for the 0-1 knapsack problem, and variations. Their most efficient algorithm utilizes a lower bound P_0, $P_0 \leq P^* \leq 2P_0$, to separate items according to profits into a class of "small" items and a class of "large" items. The problem is then solved for the large items only, with profits scaled by a suitably chosen scale factor. Feasible solutions of large items are then joined with sets of small items, and the feasible solution with the largest total profit is selected.

By noting that it is possible to obtain a bound, independent of n, on the number of large items which need to be considered for their computation, Ibarra and Kim claim a bound of

$$O(n \log n + \frac{1}{\varepsilon^4} \log(\frac{1}{\varepsilon})) \tag{1.1}$$

on running time and

$$O(n + \frac{1}{\varepsilon^3}) \tag{1.2}$$

on space requirements.

In this paper we elaborate on the Ibarra-Kim approach, introducing a number of improvements which yield better time and space bounds and also

tend to enhance the practicality of the procedures. Among the modifications proposed are the use of a median-finding routine to eliminate sorting, and a more efficient scaling technique. A number of other modifications are proposed, including alternative data structures and a different method for carrying out the large-item computation.

As a result of these changes, we are able to obtain the following bounds. For the 0-1 problem: $O(n \log(\frac{1}{\varepsilon}) + \frac{1}{\varepsilon^4})$ time and $O(n + \frac{1}{\varepsilon^3})$ space. For the unbounded problem: $O(n + \frac{1}{\varepsilon^3})$ time and space. For the subset-sum problem: $O(n + \frac{1}{\varepsilon^3})$ time and $O(n + \frac{1}{\varepsilon^2})$ space, or $O(n + \frac{1}{\varepsilon^2} \log(\frac{1}{\varepsilon}))$ time and space. The "multiple-choice" problem, with m equivalence classes, is solved in $O(\frac{nm^2}{\varepsilon})$ time and space.

The organization of this paper is as follows. In Section 2 a basic optimization procedure is described. Modifications of this procedure are presented in Sections 3-11. These yield various approximation algorithms for the 0-1 knapsack problem. In Sections 12 through 15, algorithms are outlined for related problems, including the unbounded problem, the subset-sum problem, and "multiple-choice" knapsack problems. Concluding sections, 16 and 17, indicate possible further extensions and directions for future research.

Comment: Complexity estimates are based on the assumption that computer word length is sufficient to accommodate numbers as large as P^*, b, and n. Arithmetic operations on numbers as large as these are assumed to require constant time. Thus, factors such as $\log b$ do not appear in bounds such as (1.1) and (1.2). However, we shall *not* assume word length on the order of $1/\varepsilon$ or n bits (numbers as large as $2^{1/\varepsilon}$ or 2^n).

2. A Basic Optimization Procedure

We begin with the description of an optimization algorithm for the 0-1 knapsack problem. All of the approximation algorithms presented in this paper are modifications or adaptations of this basic procedure.

One way to solve the 0-1 knapsack problem is to generate a list of all feasible combinations of profit and weight. Each such combination is represented by a pair (P,A), for which there is a subset of items S with

$$\sum_{j\in S} p_j = P ,$$

$$\sum_{j\in S} a_j = A \leq b .$$

The value of P^* is then given by a pair (P,A) for which P is maximum.

The list can be generated in n iterations as follows. Initially place only the pair $(0,0)$ in the list. Then, at iteration j, form from each pair (P,A) a "candidate" pair $(P+p_j,A+a_j)$, if $A+a_j \leq b$, and place the new pair in the list if it does not duplicate an existing one. The result is that at the end of iteration j, each pair in the list represents a feasible profit-weight combination for some subset of items $S \subseteq \{1,2,\ldots,j\}$, and each such subset is represented by a pair.

This procedure is unnecessarily inefficient, because it generates many pairs which are not needed to determine an optimal solution. It clearly does not affect the computed value of P^* if "dominated" pairs are discarded. That is, if (P,A) is in the list, one may eliminate any pair (P',A'), where $P \geq P'$, and $A \leq A'$. After dominated pairs are eliminated, each remaining pair (P,A) satisfies the following conditions at the end of iteration j: P is the largest attainable profit for a subset of items

$S \subseteq \{1,2,\ldots,j\}$, with weight at most equal to A, and A is the smallest attainable weight for a subset of items with profit at least equal to P.

The procedure is now revised as follows. At the end of each iteration, the pairs (P,A) are in strictly increasing order of P and of A. To perform iteration j, produce a candidate pair $(P+p_j, A+a_j)$ for each pair (P,A), provided $A + a_j \leq b$. Since the list is in increasing order of A, the production of candidate pairs can be terminated whenever a pair (P,A) is reached for which $A + a_j > b$. Then merge the existing list and the list of candidates, discarding dominated pairs in the process. This is easily accomplished, because of the ordering of the P's and A's. At the end of iteration n, the last pair in the list is (P^*,A^*), where A^* is the minimum attainable weight for an optimal solution.

There are various data structures and list merging techniques which are suitable for processing the list of pairs. In any case, it is clear that at each iteration the running time and space requirements are linearly proportional to the number of pairs existing in the list at the beginning of the iteration. An upper bound on the number of pairs in the list is $\min\{P^*,b\}$. Hence an upper bound on the running time for the overall procedure, in the worst case, is $O(n \min\{P^*,b\})$, and an upper bound on space requirements is $O(n + \min\{P^*,b\})$. (The term n in the space bound accounts for the storage of input data.)

Up to this point, we have ignored the problem of constructing an optimal solution, i.e. determining a set S^* corresponding to the last pair (P^*,A^*) in the list at the end of iteration n. There are at least two methods for doing this.

A very straightforward method, employed in [4], is to convert each pair (P,A) to a triple (P,A,S), where S is a list of indices of items such that

$$\sum_{j \in S} p_j = P \ , \qquad \sum_{j \in S} a_j = A \ .$$

Initially, only the triple $(0,0,0)$ is placed in the list. Thereafter, at iteration j, each candidate triple is of the form $(P+p_j, A+a_j, S \cup \{j\})$. This has the effect of increasing the space bound to $O(n \min\{P^*,b\})$, since S may be $O(n)$ in size. Forming the set $S \cup \{j\}$ may be assumed to require $O(n)$ time, thereby also adding an extra factor of n to the time bound, running it up to $O(n^2 \min\{P^*,n\})$.

Comment: It appears that the time required to form $S \cup \{j\}$ is disregarded in [4], and that consequently the time bound (1.1) should be raised to

$$O(n \log n + \frac{1}{\varepsilon^5} \log(\frac{1}{\varepsilon})) \ .$$

We choose not to provide an explicit representation of the set S corresponding to a pair (P,A). Instead, we propose to construct S by "backtracing" through a secondary data structure in the form of a rooted tree.

The necessary data structures are indicated in Figure 1. Each entry in the list of (P,A) pairs has four components: a P value, an A value, a pointer to the next entry in the list, and a pointer to a node in the rooted tree. Each tree node has two components: an item index j and a pointer to its parent node.

To find the set S for an entry (P,A) in the list, one goes to the tree node indicated by its pointer and then reads off the item indices

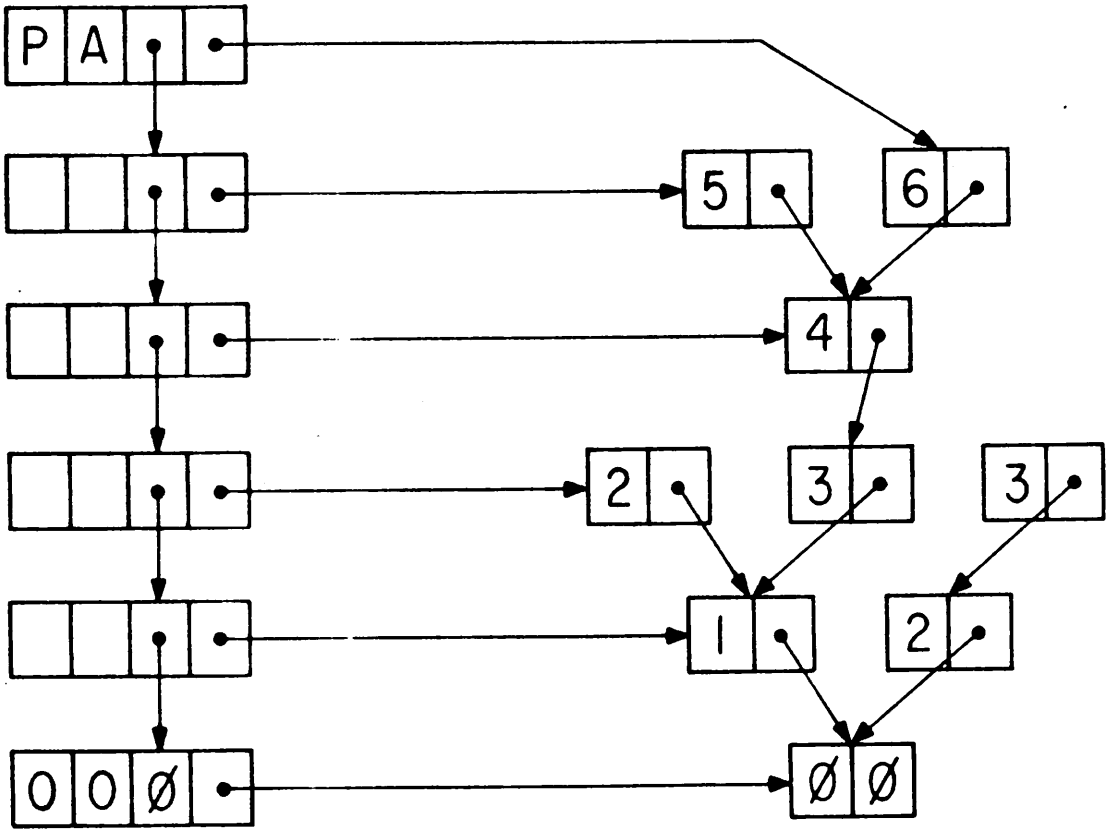

Figure 1. Data structures for list of (P,A) pairs and tree for backtracing.

associated with nodes on the path to the root. Thus, for the entry at the head of the list in Figure 1, $S = \{6,4,3,1\}$. It is easily seen that S can be constructed in $O(n)$ time.

Initially the list contains only the pair $(0,0)$ and the tree pointer for this pair is directed to the root of the tree. Thereafter, whenever a pair $(P+p_j,A+a_j)$ is added to the list of (P,A) pairs, the tree pointer for $(P+p_j,A+a_j)$ is directed to a new tree node with associated item index j. The pointer for this new node is directed to the node pointed to by (P,A). It is clear that these operations can be implemented in constant time for each new pair $(P+p_j,A+a_j)$, or in $O(n \min\{P^*,b\})$ time overall. Moreover, the tree requires $O(n \min\{P^*,b\})$ space.

3. Scaling of Profit Space

One way to make the computation more efficient is to reduce the number of distinct P (or A) values which may occur in the pairs (P,A). The simplest method to accomplish this is to replace each p_j coefficient by

$$q_j = \left\lfloor \frac{p_j}{K} \right\rfloor ,$$

where K is a suitably chosen scale factor. Then $\frac{P^*}{K}$ replaces P^* in the time and space bounds given above.

How large can we make K, and be assured that the solution we obtain differs from the optimum by no more than εP^*? We note the inequality,

$$Kq_j \leq p_j < K(q_j+1) . \tag{3.1}$$

It follows that for any set S,

$$\sum_{j \in S} p_j - K \sum_{j \in S} q_j < K|S| .$$

Hence if we can insure that

$$K|S^*| \leq \varepsilon P^* ,$$

where S^* is an optimal solution, then K will be a valid scale factor: the solution found by the computation outlined in the previous section will be within the prescribed accuracy $\varepsilon > 0$.

But surely $|S^*| < n$ and $P^* \leq p_{max}$, where

$$p_{max} = \max_j \{p_j\} .$$

(We can assume $a_j \leq b$ for all items.) Hence we may choose

$$K = \frac{1}{n} \varepsilon p_{max} . \tag{3.2}$$

Now $P^* \leq np_{max}$, so

$$\frac{P^*}{K} \leq \frac{n^2}{\varepsilon} .$$

Substituting $\frac{n^2}{\varepsilon}$ for P^* in the bounds obtained in the previous section, we obtain time and space bounds of $O(\frac{n^3}{\varepsilon})$, with assurance that the relative error of the solution we obtain does not exceed ε, as specified. (Hereafter we ignore the role of b in time and space bounds.) We have thus obtained a fully polynomial approximation algorithm.

4. A Lower Bound on P^*

It is evident that a lower bound on P^* better than p_{max} will enable us to increase the size of the scale factor K and thereby improve the efficiency of the computation.

If we relax the conditions $x_j \in \{0,1\}$ to $0 \leq x_j \leq 1$, a linear programming problem results. It can be solved quite simply: First sort the items in nonincreasing p_j/a_j ratio order, so that, without loss of generality,

$$\frac{p_1}{a_1} \geq \frac{p_2}{a_2} \geq \cdots \geq \frac{p_n}{a_n} .$$

Then place the items in the knapsack in order until either (a) the items are exhausted or (b) the capacity is exactly used up or (c) it is necessary to fractionalize one item to use up the capacity exactly. In cases (a) and (b), an optimal solution is obtained, and it is unnecessary to proceed further.

So suppose case (c) occurs and

$$a_1 + a_2 + \cdots + a_j < b$$

but

$$a_1 + a_2 + \cdots + a_j + a_{j+1} > b .$$

We assert that

$$P_0 \leq P^* \leq 2P_0 , \tag{4.1}$$

where

$$P_0 = \max\{p_1 + p_2 + \cdots + p_j,\ p_{max}\} ,$$

and where

$$p_{max} = \max_j \{p_j\} ,$$

as before. This is because

$$p_1 + p_2 + \cdots + p_j \leq P^* ,$$

$$p_{j+1} \leq p_{max} \leq P^* ,$$

but

$$p_1 + p_2 + \cdots + p_{j+1} > P^* .$$

Replacing p_{max} by P_0 in (3.2), we obtain

$$K = \frac{1}{n}\varepsilon P_0 , \tag{4.2}$$

and find that

$$\frac{P^*}{K} \leq \frac{2n}{\varepsilon} .$$

The computation can now be carried out in $O(\frac{n^2}{\varepsilon})$ time and space, exclusive of the time required to sort the items in p_j/a_j order.

But sorting is not necessary to compute P_0. This can be done in $O(n)$ time by employing a median-finding algorithm as follows: First compute the ratio p_j/a_j for all items. Then find the median of these ratios. (All ratios are considered to be distinct; if ties occur, the item with the smaller index is considered to have the smaller ratio.) Suppose the median ratio is r and let

$$J = \{j \mid p_j/a_j \geq r\} . \tag{4.3}$$

If $\sum_{j \in J} a_j > (<) b$, find the median ratio in (the complement of) J until the largest set J is found such that $\sum_{j \in J} a_j \leq b$.

Case (a) above occurs if J contains all items. Case (b) occurs if

$$\sum_{j \in J} a_j = b .$$

Otherwise, case (c) occurs and

$$P_0 = \max\{\sum_{j \in J} p_j,\ p_{max}\} .$$

There are median-finding routines which require only $O(n)$ time [1]. This procedure requires $O(\log n)$ applications of such a routine, but these are carried out over sets which contain $n, \frac{n}{2}, \frac{n}{4}, \ldots$ elements. It is thus evident that the computation of P_0 requires only $O(n)$ time and space.

In the next section, we shall have need for a procedure which will fill the knapsack to any desired capacity b', $0 \leq b' \leq b$, just as we filled it to capacity b in computing P_0. This means finding the largest ratio r such that $\sum_{j \in J} a_j \leq b'$, where J is defined as in (4.3). We shall let $\phi(b')$ denote the total profit of the items so placed in the knapsack:

$$\phi(b') = \sum_{j \in J} p_j .$$

5. Separation of Items

The existence of the lower bound P_0 enabled us to reduce the time bound from $O(\frac{n^3}{\varepsilon})$ to $O(\frac{n^2}{\varepsilon})$. A technique of Ibarra and Kim enables us to reduce the bound still further, to $O(\frac{n}{\varepsilon^2})$.

Comment: The reader may question whether $\frac{n}{\varepsilon^2}$ is "better" than $\frac{n^2}{\varepsilon}$. The bounds stated are intended to emphasize asymptotic behavior in n, rather than ε. The algorithm we are about to describe will, in fact, be bounded by $O(\frac{n^2}{\varepsilon})$, as well as $O(\frac{n}{\varepsilon^2})$.

The Ibarra-Kim approach is as follows: First compute P_0, as described in the previous section. Use P_0 to determine a threshold value T. Items with $p_j \leq T$ are considered "small," and those with $p_j > T$ "large." Solve the problem, using the large items only, with some appropriately chosen scale factor K. This yields a final list of (Q,A) pairs, where Q denotes total scaled profit.

For each pair (Q,A) in the final list, fill in the remaining knapsack capacity b-A with small items, as indicated in the previous section. These small items yield total profit $\phi(b-A)$. The approximate solution, a combination of large and small items, is chosen to yield profit P, where

$$P = \max_{(Q,A)} \{KQ + \phi(b-A)\} . \qquad (5.1)$$

Intuitively, it seems reasonable to divide the permissible error equally between the small items and the large items. This suggests setting $T = \frac{1}{2}\varepsilon P_0$, which has the following result: When space b-A is filled with small items, at most one item with profit no greater than $\frac{1}{2}\varepsilon P_0$ (half the estimated permissible error) will be omitted. The scale factor K should then be chosen so that the total error contributed by the large items is also no more than one half the permissible amount. Below we justify this intuitive argument, and prove that this choice of T also enables us to maximize the scale factor K.

Suppose there exists an optimal solution in which the large items contribute profit P_L and weight A_L and the small items P_S and A_S. Let Q_L be the sum of the scaled profit coefficients contributing to P_L, using scale factor K. It should be apparent, without need of proof, that the

final list for the large-item computation must contain a pair (Q,A), dominating (Q_L,A_L), i.e.,

$$Q \geq Q_L , \qquad A \leq A_L .$$

Thus P, determined by (5.1), must be such that

$$P \geq KQ + \phi(b-A) \geq KQ_L + \phi(b-A) . \tag{5.2}$$

The number of large items contributing to the optimal solution cannot exceed P_L/T, where

$$\frac{P_L}{T} \leq \frac{P^*}{T} . \tag{5.3}$$

Employing the inequality (3.1),

$$Kq_j \leq p_j < K(q_j+1) ,$$

with (5.3), we obtain

$$P^* = P_L + P_S < K(Q_L + \frac{P^*}{T}) + P_S . \tag{5.4}$$

From (5.2) and (5.4) it follows that

$$P^* - P < \frac{K}{T}P^* + P_S - \phi(b-A) . \tag{5.5}$$

But $b-A \geq b-A_s$. If our procedure exactly filled the capacity $b-A$ with small items, then $\phi(b-A) \geq P_S$. If it left some unused capacity, the profit of the item which could not be fit in was no greater than T, and

$$\phi(b-A) + T \geq P_S . \tag{5.6}$$

From inequalities (5.5) and (5.6), we obtain

$$P^* - P < \frac{K}{T}P^* + T \ . \tag{5.7}$$

It follows that T and K should be chosen to insure that

$$\frac{K}{T}P^* + T \leq \varepsilon P^* \ ,$$

which can be done by letting

$$\frac{K}{T} = \lambda\varepsilon \ ,$$

and

$$T = (1-\lambda)\varepsilon P_0 \leq (1-\lambda)\varepsilon P^* \ ,$$

for any λ, $0 \leq \lambda < 1$. Assuming $T \neq 0$, this means

$$K = \lambda(1-\lambda)\varepsilon^2 P_0 \ .$$

Since we wish to maximize K, we choose $\lambda = \frac{1}{2}$, yielding

$$K = \frac{1}{4}\varepsilon^2 P_0 \ , \qquad T = \frac{1}{2}\varepsilon P_0 \ . \tag{5.8}$$

This confirms our intuition as to the correct choice of T and K.

Comment: In [4], the choice of K and T was, in effect:

$$K = \frac{2}{9}\varepsilon^2 P_0 \ , \qquad T = \frac{2}{3}\varepsilon P_0 \ ,$$

corresponding to a choice of $\lambda = \frac{1}{3}$.

Observe that

$$\left\lceil \frac{P^*}{K} \right\rceil \leq \frac{2P_0}{\frac{1}{4}\varepsilon^2 P_0} = \frac{8}{\varepsilon^2} \ ,$$

so the size of scaled profit space is not $O(\frac{1}{\varepsilon^2})$, instead of $O(\frac{n}{\varepsilon})$. Time

and space for the large-item computation are bounded by $O(\frac{n}{\epsilon^2})$, since there are n iterations.

We have yet to discuss the computation of the ϕ-values at the end of the large-item computation. This can be accomplished in $O(n \log(\frac{1}{\epsilon}))$ time, as we shall show in the next section. Hence the overall time and space bounds for the procedure are $O(\frac{n}{\epsilon^2})$. Now notice that if $T < 1$ (all items "large") as determined by (5.8), then certainly $K < 1$ in (4,2). Hence there is no instance in which the method of the previous section provides a useful scale factor (greater than unity) and the present one does not. Moreover, if

$$T = \frac{1}{2}\epsilon P_0 < 1 ,$$

then one should simply solve the problem optimally, which can be done in $O(\frac{n}{\epsilon})$ time, since $P^* < \frac{4}{\epsilon}$.

Next notice that P^*/T can be replaced in (5.5) by n, yielding

$$P^* - P < Kn + T .$$

Assuming $\lambda = \frac{1}{2}$, this implies that K should be of the form

$$K = \frac{1}{2n}\epsilon P_0 ,$$

which results in

$$\frac{P^*}{K} \leq \frac{4n}{\epsilon}$$

and an $O(\frac{n^2}{\epsilon})$ computation, as in Section 4.

There observations suggest that equations (5.8) should be modified to become

$$\left.\begin{aligned} K &= \max\{\tfrac{1}{4}\varepsilon^2 P_0,\ \tfrac{1}{2n}\varepsilon P_0,\ 1\}\ , \\ T &= \begin{cases} \varepsilon P_0 & \text{if } K = 1\ , \\ \tfrac{1}{2}\varepsilon P_0 & \text{if } K > 1\ . \end{cases} \end{aligned}\right\} \tag{5.9}$$

Equations (5.9) assure a computation which is time and space bounded by both $O(\frac{n^2}{\varepsilon})$ and $O(\frac{n}{\varepsilon^2})$. Moreover, whenever $T < 1$ in (5.9), an *optimal* solution is obtained in $O(\frac{n}{\varepsilon})$ time. Although in succeeding sections we shall present ideas leading to substantially better asymptotic performance with respect to n, we shall not be able to improve on $O(\frac{n^2}{\varepsilon})$, for asymptotic performance with respect to ε.

6. Computation of ϕ-Values

It is easy to compute the ϕ-values in $O(n+\frac{1}{\varepsilon^2})$ time, as in [4], once the small items have been placed in p_j/a_j ratio order. However, we eliminated sorting in the computation of P_0, so we do not have an ordered set of small items available as a byproduct. The procedure we propose is as follows.

First find the median p_j/a_j ratio for the set of all small items. Let this ratio be r and let

$$J = \{j \mid p_j/a_j \geq r\}\ ,$$

as before. If $\sum_{j\in J} a_j > b$, throw away the complement of J and repeat, continuing to throw away half-sets until a ratio r is found, such that $\sum_{j\in J} a_j \leq b$. Then search the list of pairs until a pair $(\bar{Q},\bar{A})$ is found, such that

$$\bar{A} = \max_{(Q,A)} \{A \mid \sum_{j \in J} a_j \leq b-A\} ,$$

and

$$\phi(b-\bar{A}) = \sum_{j \in J} p_j .$$

At this point one ϕ-value has been found in $O(n)$ time, exclusive of the time required to find the pair $(\bar{Q},\bar{A})$. A set J, $|J| \leq \frac{n}{2}$, has been determined, as has a complementary set $\bar{J}$, $|\bar{J}| \leq \frac{n}{2}$, within the set of remaining small items. Proper subsets of J determine $\phi(b,A)$ for $A > \bar{A}$. Proper subsets of $\bar{J}$, joined with all the items in J, determine $\phi(b-A)$ for $A < \bar{A}$. Thus there are now two subproblems of the same character as the original one.

The difficulty in estimating the remaining time required for the procedure is that we cannot be sure how many pairs are involved for each subproblem: about $\frac{4}{\varepsilon^2}$ for each or $\frac{8}{\varepsilon^2}$ for one and none for the other? We can confirm that time is bounded by $O(n \log(\frac{1}{\varepsilon}))$ by the following analysis:

Let

$T(m,n)$ = the time required for median finding, given m pairs and n small items.

Then

$$T(m,n) = cn + \max_{1 \leq q \leq m} \{T(m-q,\tfrac{n}{2}) + T(q-1,\tfrac{n}{2})\} ,$$

where c is a constant. Assume $T(m,n) \leq cn \log_2 m$. Then

$$\begin{aligned} T(m,n) &\leq cn + \max_q \{c(\tfrac{n}{2})[\log_2(m-q) + \log_2(q-1)]\} \\ &\leq cn + cn \log_2(\tfrac{m-1}{2}) \\ &\leq cn \log_2 m , \end{aligned}$$

as required.

Thus, median-finding requires at most $O(n \log(\frac{1}{\varepsilon}))$ time, since m is $O(\frac{1}{\varepsilon^2})$. Other operations required in determining the ϕ-values (such as locating pairs $(\bar{Q},\bar{A})$) are bounded by $O(\frac{1}{\varepsilon^2})$.

7. Discarding Superfluous Items

The time and space bounds can be further reduced by the following observations.

The number of distinct q_j values that can exist for the large items is bounded by

$$\left\lceil \frac{P^*}{K} \right\rceil - \left\lceil \frac{T}{K} \right\rceil \leq \frac{8}{\varepsilon^2} - \frac{2}{\varepsilon} .$$

The number of large items with scaled profit q_j which can be contained in any feasible solution is at most

$$n_j = \left\lfloor \frac{P^*}{Kq_j} \right\rfloor .$$

Hence for any scaled profit q_j, one need only consider the n_j items with smallest weight for use in the large-item computation.

For q_j values in the interval $(\frac{2}{\varepsilon},\frac{4}{\varepsilon}]$, the average value of n_j is about $\frac{3}{\varepsilon}$. For the interval $(\frac{4}{\varepsilon},\frac{8}{\varepsilon}]$, the average value of n_j is about $\frac{3}{2\varepsilon}$. For each successive interval, the average value of n_j is half as large, but the interval contains twice as many q_j values. There are at most $\log_2(\frac{P^*}{T}) \leq \log_2(\frac{4}{\varepsilon})$ intervals. Hence the number of items which must be considered for the large-item computation is bounded by

$$\frac{6}{\varepsilon^2} \log_2(\frac{4}{\varepsilon}) . \qquad (7.1)$$

Identifying the items which need be considered for the large-item computation is simple: Place the large items in at most $\max\{n,\frac{8}{\epsilon^2}\}$ buckets, each bucket containing items with the same q_j value. Then apply a median-finding routine to each bucket q_j, to identify the n_j items with smallest weight. This can be done in $O(n)$ time.

We can now substitute $\frac{1}{\epsilon^2}\log(\frac{1}{\epsilon})$ for n in the time and space bounds for the large-item computation obtained in Section 5. This yields a time bound of $O(n\log(\frac{1}{\epsilon}) + \frac{1}{\epsilon^4}\log(\frac{1}{\epsilon}))$ and a space bound of $O(n+\frac{1}{\epsilon^4}\log(\frac{1}{\epsilon}))$ for the overall computation. (Note that $O(n)$ space is required to store the input data.)

8. An Improved Method of Scaling

The method of scaling we have employed up to this point is unnecessarily conservative. The same fixed error, K, has been permitted each scaled profit coefficient q_j. It is more effective to produce coefficients for which the permissible error is somewhat in proportion to the size of the coefficient. This can be done as follows.

Let

$$T = \frac{1}{2}\epsilon P_0 , \qquad K = \frac{1}{2}\epsilon T = \frac{1}{4}\epsilon^2 P_0 , \tag{8.1}$$

as in Section 5. If p_j lies in the interval $(2^k T, 2^{k+1} T]$, $k = 0,1,2,\ldots,\lfloor\log_2\frac{P^*}{T}\rfloor$, then let

$$q_j = \left\lfloor \frac{p_j}{2^k K} \right\rfloor 2^k . \tag{8.2}$$

Now notice that

$$\begin{aligned} Kq_j \le p_j &< Kq_j + 2^k K \\ &\le Kq_j + \tfrac{1}{2}\epsilon 2^k T \\ &\le Kq_j + \tfrac{1}{2}\epsilon p_j \ . \end{aligned}$$

The above inequality is parallel to (3.1). It follows that we have, in place of inequality (5.4),

$$P^* = P_L + P_S < KQ_L + \tfrac{1}{2}\epsilon P_L + P_S \ . \tag{8.3}$$

By reasoning similar to that used before, we obtain

$$P^* - P < \tfrac{1}{2}\epsilon P^* + T = \tfrac{1}{2}\epsilon P^* + \tfrac{1}{2}\epsilon P_0 \ . \tag{8.4}$$

Let $m = \lfloor \log_2 \frac{P^*}{T} \rfloor$. The size of scaled profit space is bounded by

$$\left\lfloor \frac{P^*}{2^m K} \right\rfloor 2^m \le \frac{P^*}{K} \le \frac{8}{\epsilon^2} \ ,$$

as before. However, the number of distinct q_j values obtained from each p_j-interval $(2^k T, 2^{k+1} T]$ is at most $\frac{2}{\epsilon} - 1$, independent of k. Thus for the scaled profit-space interval $(\frac{2}{\epsilon}, \frac{4}{\epsilon}]$, the number of q_j values is about $\frac{2}{\epsilon}$, and the average value of n_j is $\frac{3}{\epsilon}$. For the interval $(\frac{4}{\epsilon}, \frac{8}{\epsilon}]$, the number of q_j values is still $\frac{2}{\epsilon}$, and the average value of n_j is $\frac{3}{2\epsilon}$, and so on. It follows that the number of items which must be considered for the large-item computation is bounded by

$$\frac{12}{\epsilon^2}\left(1 + \frac{1}{2} + \cdots + \frac{T}{P^*}\right) \le \frac{12}{\epsilon^2} \ .$$

We have thus eliminated the log factor in (7.1).

Comment: We have dispensed with a more general argument, in which $T = (1-\lambda)\varepsilon P_0$, $K = \lambda\varepsilon T$, as this carries through exactly as in Section 5. Nor shall we confirm that there is no advantage in choosing an integer $a > 2$, and letting

$$q_j = \frac{p_j}{\lfloor a^k K \rfloor} a^k ,$$

if p_j lies in the interval $(a^k T, a^{k+1} T]$. Note that the previous scaling technique is the case in which $a \geq P^*/T$.

Since there are now $O(\frac{1}{\varepsilon^2})$ items for the large-item computation, the time bound can be reduced to $O(n \log(\frac{1}{\varepsilon}) + \frac{1}{\varepsilon^4})$ and the space bound to $O(n + \frac{1}{\varepsilon^4})$. In the next section we shall show how the space bound can be further reduced to $O(n + \frac{1}{\varepsilon^3})$, while maintaining the time bound.

9. Modification of the Large-Item Computation

We propose to modify the large-item computation to reduce the space required for backtracing. Our plan is as follows.

First sort at most n_j items with a given q_j value into nondecreasing order of weight. This can be done, for all q_j values, in $O(\frac{1}{\varepsilon^2} \log(\frac{1}{\varepsilon}))$ time. Now notice that there are at most $n_j + 1$ possibilities for each q_j value: Either no items are chosen, the first (smallest-weight) item is chosen, the first two items are chosen, ... , or all (at most n_j) items are chosen. The large-item computation is now carried out in iterations over distinct q_j values (instead of individual items) in strictly decreasing order of q_j.

Initially the list contains only the pair (0,0). At the end of iteration i, each pair (Q,A) indicates the smallest weight A attainable for a subset of items chosen from the i largest q_j values, with total profit at least Q. It also indicates the largest profit Q attainable for a subset of items chosen from the i largest q_j values, with total weight not exceeding A.

Suppose iteration i is for scaled profit q_j and there are n_j items with this scaled profit. For each pair (Q,A) in the list, n_j candidate pairs are formed, corresponding to the choice of 1 item, 2 items, ..., n_j items. These pairs are placed in n_j separate candidate lists. The n_j+1 lists are then merged, deleting dominated pairs in the process. The merge can be carried out in $O(N_j \log n_j)$ time, where N_j is the length of the original list. This time is dominated by the $O(N_j n_j)$ time required to prepare the candidate lists. It is clear that $O(\frac{1}{\varepsilon^4})$ time is sufficient for this procedure, since $\sum n_j \leq \frac{12}{\varepsilon^2}$ and $N_j \leq \frac{8}{\varepsilon^2}$ at each iteration.

Now let us consider the space bound. At each iteration, the space required for candidate lists is at most $n_j N_j$, which is $O(\frac{1}{\varepsilon^3})$. The number of new entries added to the list of pairs, and hence the number of nodes added to the tree used for backtracing is $O(N_j)$. But N_j is at most 1 for the largest interval, 2 for the second-largest, ... , $\frac{2}{\varepsilon^2}$ for the second-smallest, and $\frac{4}{\varepsilon^2}$ for the smallest. This is because the q_j values in each interval are a power of 2, which reduces the effective size of the scaled profit space from $\frac{P^*}{K} \leq \frac{8}{\varepsilon^2}$ to $2^{-k}\frac{P^*}{K} \leq 2^{-k}(\frac{8}{\varepsilon^2})$, for iterations over q_j values in interval k. The number of q_j values (iterations) for each interval is at most $T \leq \frac{2}{\varepsilon}$. It follows that the total number of nodes in the tree is bounded by a number proportional to

$$\frac{16}{\varepsilon^3}[1+\frac{1}{4}+\frac{1}{16}+\cdots] < \frac{22}{\varepsilon^3} . \qquad (9.1)$$

Hence total space is bounded by $O(n+\frac{1}{\varepsilon^3})$.

Comment: Each of the n_j candidate pairs obtained from a pair (Q,A) can be viewed as corresponding to the choice of a "multiple" item. The indexing scheme and backtracing procedure must be slightly modified. It is not difficult to do this, while staying within the time and space bounds. Hereafter, we shall not mention necessary modifications of the indexing scheme and backtracing procedure, assuming the reader can supply details.

10. Summary of Algorithm

We now summarize the steps of the approximation algorithm for the 0-1 knapsack problem:

1. Compute p_j/a_j ratios for all items. Compute P_0, using a median-finding routine as indicated in Section 4: $O(n)$ time and space.
2. Determine the threshold T and scale factor K by (8.1). Compute q_j for all large items, using equation (8.2): $O(n)$ time and space.
3. Determine the $O(\frac{1}{\varepsilon^2})$ items to enter into the large-item computation, by applying a median-finding routine to find the at most n_j items with smallest weight, for each q_j value: $O(n)$ time and space.
4. Sort the at most n_j items for each q_j value in order of nonincreasing weight: $O(\frac{1}{\varepsilon^2}\log(\frac{1}{\varepsilon}))$ time and $O(\frac{1}{\varepsilon^2})$ space.
5. Carry out the modified version of the large-item computation described in the previous section: $O(\frac{1}{\varepsilon^4})$ time and $O(\frac{1}{\varepsilon^3})$ space.
6. Compute $\phi(b,A)$ for each pair (Q,A) in the final list, employing a median-finding procedure, as described in Section 6: $O(n\log(\frac{1}{\varepsilon})+\frac{1}{\varepsilon^2})$ time and $O(n+\frac{1}{\varepsilon^2})$ space.

7. Find a pair (Q,A) for which $KQ+\phi(b-A)$ is maximum: $O(\frac{1}{\varepsilon^2})$ time and space.

8. Find the set of large items in the approximate solution by backtracing: $O(\frac{1}{\varepsilon})$ time and space. The set of small items in the approximate solution is readily available as a byproduct of Step 6.

In succeeding sections we shall indicate how the steps of the algorithm should be modified, for other versions of the knapsack problem.

11. "Bootstrapping" the Algorithm

A further analysis of the space bound shows that space is bounded by a constant times

$$\frac{P^*}{\varepsilon^3 P_0} .$$

Similarly, time is bounded by a constant times

$$\frac{(P^*)^2}{\varepsilon^4 P_0^2} .$$

It follows that an improvement in the quality of the lower bound P_0 will yield a reduction in the bounds by at least a linear scale factor.

One way to obtain a better lower bound is to use the algorithm itself to produce approximate solutions which provide better lower bounds. A "bootstrapping" procedure is as follows. Begin with the lower bound P_0 with accuracy $\varepsilon_0 \leq \frac{1}{2}$. Use P_0 to obtain a threshold T_1 and scale factor K_1 for accuracy $\varepsilon_1 < \varepsilon_0$. Apply the approximation algorithm to obtain an approximate solution with profit P_1 and relative error ε_1. Proceed through successive iterations, with accuracies $\varepsilon_0 > \varepsilon_1 > \varepsilon_2 > \cdots > \varepsilon_N = \varepsilon$.

At successive iterations, the lower bounds P_i, thresholds T_i and scale factors K_i are determined by the relations:

$$P_i \geq (1-\varepsilon_{i-1})P^* ,$$
$$T_i = \frac{1}{2}\varepsilon_i P_{i-1} \geq \frac{1}{2}\varepsilon_i(1-\varepsilon_{i-1})P^* ,$$
$$K_i = \frac{1}{4}\varepsilon_i^2 T_i \geq \frac{1}{4}\varepsilon_i^2(1-\varepsilon_{i-1})P^* .$$

The running time at iteration i is then bounded by a constant times

$$\frac{(P^*)^2}{\varepsilon_i^4 P_{i-1}^2} \leq \frac{P^*}{\varepsilon_i^4(1-\varepsilon_{i-1})^2} . \tag{10.1}$$

If we perform one iteration, as in the preceding, with $\varepsilon_0 = \frac{1}{2}$, $\varepsilon_1 = \varepsilon$, then the quantity (10.1) becomes

$$\frac{4P^*}{\varepsilon^4} .$$

If we perform N iterations, even if $\varepsilon_{N-1} = 0$, the quantity (10.1) becomes

$$\frac{P^*}{\varepsilon^4}$$

for the last iteration, with $\varepsilon_N = \varepsilon$. It follows that no matter how such a bootstrapping scheme is arranged, we can expect to improve the time bound by no more than a linear scale factor less than four.

The practical implications are more favorable, however. Suppose $T < 1$ and $K < 1$, as determined by (8.1). One can choose an ε_1 such that T_1 and K_1 have favorably large values. (It is unlikely, for example, that $T_1 < 1$ when $T_1 = \frac{1}{8}P_0 \geq \frac{1}{4}P^*$, for $\varepsilon_1 = \frac{1}{4}$; if this is so, then surely the problem is an easy one.) This enables the iterative process to begin, and if one is fortunate, a bound P_1, numerically much larger than P_0, will be produced.

Another advantage of iteration is that early iterations are likely, in practice, to be quite short, lengthening considerably with each successive iteration. This enables one to halt the procedure, when desired, with an approximate solution known to have accuracy ε_i, where i was the last iteration.

12. The Unbounded Problem

Recall the unbounded knapsack problem is the case in which the variables x_j are not restricted to 0,1 values, but may be nonnegative integers. A number of simplifications can be made in this case.

First, it is evident that the computation of P_0 and of the ϕ-values is now much more straightforward: One need only identify a single item with maximum p_j/a_j ratio, which can be done with a single scan through the items. A small item with maximum p_j/a_j ratio is all that is needed to compute all the ϕ-values. This can be done in $O(n+\frac{1}{\varepsilon^2})$ time.

It is evident we need retain only one large item for each q_j value for the large-item computation, i.e. one with minimum weight. However, we must provide for all possible n_j multiplicities of each such item, where

$$n_j = \left\lfloor \frac{p^*}{Kq_j} \right\rfloor \leq \left\lfloor \frac{4}{\varepsilon} \right\rfloor \quad .$$

Ibarra and Kim propose doing this by providing n_j identical copies of the item. A more sensible procedure is to provide only $\lfloor \log_2 n_j \rfloor$ additional copies by "doubling." That is, let the i^{th} copy of item j be such that

$$p_j^{(i)} = 2^i p_j \quad , \qquad a_j^{(i)} = 2^i a_j \quad .$$

All necessary copies of items can be found in $O(\frac{1}{\varepsilon^2}\log(\frac{1}{\varepsilon}))$ time. It is now necessary to retain only the smallest-weight items, or copy of an item, for each q_j value. This leaves $O(\frac{1}{\varepsilon}\log(\frac{1}{\varepsilon}))$ items for the large-item computation.

The large-item computation now proceeds by iteration over items or q_j values (there is no difference), from largest to smallest. It is evident that this can be carried out in $O(\frac{1}{\varepsilon^3})$ time and space. Hence we conclude that the unbounded knapsack problem can be solved in $O(n+\frac{1}{\varepsilon^3})$ time and space.

13. "Subset-Sum" Problem

The "subset-sum" problem is as follows: Given n positive integers $p_1, p_2, \cdots, p_n$, and an integer b, does there exist a subset S, such that

$$\sum_{j \in S} p_j = b \ ?$$

This can obviously be reduced to a 0-1 knapsack problem of the form

$$\begin{aligned} &\text{maximize} && \textstyle\sum_j p_j x_j \\ &\text{subject to} && \textstyle\sum_j p_j x_j \leq b \end{aligned}$$

$$x_j \in \{0,1\} \ .$$

This is the type of problem for which we propose to find an approximate solution with accuracy $\varepsilon > 0$.

We first observe that it is a simple matter to compute P_0. The knapsack may be filled with items in arbitrary order. This clearly requires only $O(n)$ time.

We next observe that it is possible to carry out the large-item computation without consideration of item weights. That is, it is possible to process only lists of scaled profits Q, $Q \leq \frac{b}{K}$, instead of pairs (Q,A). However, we must insure that each entry in our lists is feasible. That is, if S is the set corresponding to Q, where $KQ \leq b$, then

$$\sum_{j \in S} p_j \leq b \ .$$

In order to guarantee this, we propose to round-up in scaling, rather than rounding down, i.e. replace (8.2) by

$$q_j = \left\lceil \frac{p_j}{2^k K} \right\rceil 2^k \ .$$

When this is done, all of the preceding error analysis goes through as before.

Notice that no existing list entry need ever be eliminated by dominance. Hence the data structure used for backtracing requires only $O(\frac{1}{\varepsilon^2})$ space. (In fact, the secondary data structure can be dispensed with entirely, and the pointers replaced by pointers to other list entries.)

Let us now make a selection of items for the large-time computation. First place the large items in buckets, according to their scaled profits q_j and eliminating any surplus items (more than n_j) in any bucket. This leaves at most $O(\frac{1}{\varepsilon^2})$ items. The situation now differs from the ordinary 0-1 problem in that the items in each bucket are indistinguishable: they can all be considered to have the same weight. We now want to provide for the choice of any possible number of the items in any bucket. The procedure is as follows.

Start with the smallest and work upward. If bucket q_j contains an odd number of items, say $2k+1$, place k "multiple" items, each with scaled profit $2q_j$, in bucket $2q_j$, discarding any extra items if the capacity of bucket $2q_j$ is exceeded. This leaves one item in bucket q_j. If bucket q is nonempty and contains $2k+2$ items, do the same thing, leaving two items. This process requires $O(\frac{1}{\varepsilon^2})$ time, for all buckets.

We are now left with at most two items with any given scaled profit q_j. The large-item computation can be carried out in a straightforward fashion, in $O(\frac{1}{\varepsilon^3})$ time and $O(\frac{1}{\varepsilon^2})$ space. This yields overall time and space bounds of $O(n+\frac{1}{\varepsilon^3})$ and $O(n+\frac{1}{\varepsilon^2})$.

It is also possible to solve the subset-sum problem in $O(n+\frac{1}{\varepsilon^2}\log(\frac{1}{\varepsilon}))$ time and space by applying the computation proposed by Karp [5] to the $O(\frac{1}{\varepsilon}\log(\frac{1}{\varepsilon}))$ large items. This computation involves the consideration of "intervals" of attainable P-values. We shall not give details of this computation, referring the reader to the reference.

14. Multiple-Choice Problems

Suppose the n items are partitioned into m equivalence classes and it is stipulated that no more than one item (or multiples of one item) may be chosen from each equivalence class. Such a problem is sometimes called a *multiple-choice* knapsack problem.

The author has developed an approximation algorithm for the unbounded multiple-choice problem with time and space bounds of $O(n+\frac{1}{\varepsilon^6}\log(\frac{1}{\varepsilon}))$ and $O(n+\frac{1}{\varepsilon^4}\log(\frac{1}{\varepsilon}))$. However, this algorithm is rather complicated and very likely can be improved upon. Hence we shall limit our discussion to the 0-1 multiple choice problem.

Our first observation is that there seems to be no feasible method to compute the lower bound P_0 for the 0-1 multiple-choice problem. One can easily find an item with maximum p_j/a_j ratio in each of the m equivalence classes. But what if these m items do not fill the knapsack to capacity? There are similar, but even more severe difficulties in computing ϕ-values. There seems to be no alternative to returning to the approach of Section 3.

In order to find some sort of lower bound, find an item with maximum profit in each equivalence class. Then fill the knapsack with these m items, in nondecreasing order of profit, until either the items are exhausted or it is not possible to insert another item. (This can be done in $O(m)$ time, using a median-finding routine.)

In the former case, an optimal solution has been found, and there is nothing more to do. In the latter case, m' items have been used, where $1 \leq m' < m$. Let the total profit of these m' items be P'. Then

$$P' \leq P^* \leq \frac{m}{m'}P' \leq mP' .$$

We now propose to let P' play the same role as p_{max} in the approach of Section 3. Thus, we let

$$K = \frac{m'}{m}\epsilon P' \leq \frac{1}{m}\epsilon P' .$$

Note that the size of scaled profit space is given by

$$\frac{P^*}{K} \leq \frac{m^2}{\epsilon} .$$

The list of pairs (Q,A) is processed with iteration over equivalence classes, rather than single items. The procedure is very similar to that

proposed for the large-item computation in Section 9.

Initially the list contains only the pair (0,0). At the end of iteration i, each pair (Q,A) is identified with a feasible solution containing items chosen from equivalence classes 1 through i. Suppose there are n_i items in equivalence class i. To perform iteration i, form n_i candidate items for each pair (Q,A) existing in the list at the end of iteration i-1. These candidate pairs are placed in n_i separate candidate lists. The n_i+1 lists are then merged, eliminating dominated entries.

Iteration i requires $O(\frac{n_i m^2}{\varepsilon})$ time and space, and at most $O(\frac{m^2}{\varepsilon})$ nodes are added to the tree used for backtracing. Hence overall time and space requirements are bounded by $O(\frac{nm^2}{\varepsilon})$.

Comment: The number of items which need be considered from each equivalence class is bounded by the number of distinct q_j values, which is $O(\frac{m^2}{\varepsilon})$. Hence time and space bounds of $O(n+\frac{m^5}{\varepsilon^2})$ and $O(n+\frac{m^4}{\varepsilon^2})$ can also be obtained.

15. Separable Nonlinear Functions

One considerable generalization of the knapsack problem is:

$$\text{maximize} \quad \sum_{j=1}^{m} p_j(x_j)$$

$$\text{subject to} \quad \sum_{j=1}^{m} a_j(x_j) \leq b \ ,$$

$$x_j \text{ nonnegative integer,}$$

$$x_j \leq n_j \ .$$

Here p_j and a_j are arbitrary real-valued functions, $j = 1,2,\ldots,m$.

By evaluating each function at each feasible integer point, one obtains $n = \sum n_j$ items for a 0-1 multiple-choice knapsack problem, with m equivalence classes. This can be solved, for any prescribed relative error $\varepsilon > 0$, in $O(\frac{nm^2}{\varepsilon})$ time and space, using the procedure of the previous section.

Note that there is nothing in our theory which cannot accommodate real-valued profits and weights. (Of course, items with negative profits or weights may be neglected, or may cause a knapsack problem to become unbounded, under certain obvious conditions.)

16. Further Extensions

The knapsack problem arises in many applications. However, it is a greatly specialized version of more general integer programming models for which there is a real need for approximation algorithms.

Certainly the techniques discussed in this paper fall far short of providing a fully polynomial approximation algorithm for multi-constraint problems. The principal reason is that they involve rescaling profit (objective) space instead of weight (constraint) space. Until a new technique is devised, there seems to be no reasonable way to apply the present approach to multi-constraint problems.

That is, unless we are willing to modify our views of optimization and approximation. For example, it is clearly possible to scale weight space to obtain an approximate knapsack solution of the following type: Given $\delta > 0$, find a subset of items S, such that

$$\sum_{j \in S} p_j \geq P^*$$

and

$$\sum_{j \in S} a_j \leq (1+\delta)b \ .$$

At first glance, the above may seem like an unnatural form of approximation. Possibly this because we have been taught that constraints are inviolate. But suppose the knapsack problem is being used as a simplified model for, say, project scheduling. A manager seeks to choose projects for a certain period, subject to certain resource constraints (knapsack capacity). The profits associated with the items are real and hard. The constraints are soft and flexible. He certainly wants to earn P^* dollars, if possible. Which type of approximation is more reasonable?

If the notion of constraint approximation is accepted, then it seems feasible to move ahead with the application of known techniques to multi-constraint problems.

17. Concluding Remarks

It is certainly possible that the time and space bounds presented here can be improved upon. Aside from improvements in factors of $\frac{1}{\epsilon}$, or $\log(\frac{1}{\epsilon})$, there are a number of open questions and directions for future research which suggest themselves.

We have made a few simple assumptions in making time and space bounds. Among these are that arithmetic operations can be performed in constant time on integer operands as large as P^* and b. In the case of the 0-1 knapsack problem, operations on integers as large as n are assumed, for the purpose of finding medians. Can this assumption be removed?

Perhaps a more interesting question is: Can a 0-1 approximation algorithm be found which is "strictly linear" in n, instead of order $n \log(\frac{1}{\varepsilon})$? More generally, is it possible to establish that the 0-1 problem is inherently more complex than the unbounded problem?

Acknowledgment

The author wishes to thank Alexander Rinnooy Kan and David Lichtenstein for reading preliminary drafts of this paper, and for their helpful comments and suggestions.

REFERENCES

[1] M. Blum, R.W. Floyd, V. Pratt, R.L. Rivest and R.E. Tarjan, "Time bounds for selection," *J. Computer and System Sciences* 7 (1973) 448-461.

[2] S.E. Dreyfus and R.E. Bellman, *Applied Dynamic Programming*, Princeton University Press, 1962.

[3] M.R. Garey and D.S. Johnson, "Strong NP-completeness results: motivation, examples and implications," submitted for publication.

[4] O.H. Ibarra and C.E. Kim, "Fast approximation algorithms for the knapsack and sum of subset problems," *JACM* 22 (1975) 463-468.

[5] R.M. Karp, "The fast approximate solution of hard combinatorial problems," *Proc. 6th Southeastern Conference on Combinatorics, Graph Theory, and Computing*, Utilitas Mathematica Publishing, Winnipeg, 1975, pp. 15-31.