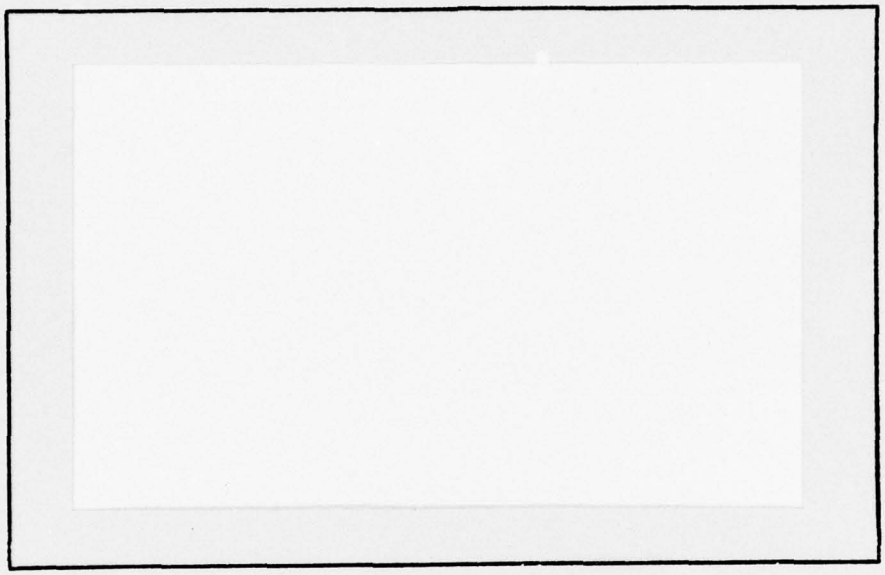
AD-A045 529 CARNEGIE-MELLON UNIV PITTSBURGH PA MANAGEMENT SCIENC--ETC F/G 12/1
SOLVING LARGE ZERO-ONE KNAPSACK PROBLEMS.(U)
JUL 77 E BALAS, E ZEMEL N00014-75-C-0621
UNCLASSIFIED RR-408 NL

1 OF 1
ADA045 529

END
DATE FILMED
11-77
DDC

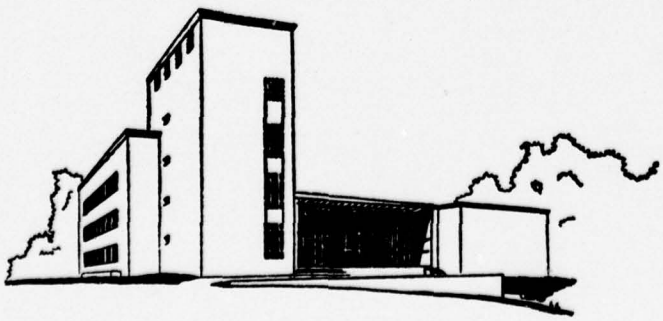


Carnegie-Mellon University

PITTSBURGH, PENNSYLVANIA 15213

GRADUATE SCHOOL OF INDUSTRIAL ADMINISTRATION

WILLIAM LARIMER MELLON, FOUNDER



(9) Management Sciences Research Report No. 408

(6) SOLVING LARGE ZERO-ONE KNAPSACK PROBLEMS.

(14) MRR-408

by

(10) Egon Balas and Eitan Zemel*

June 1976

Revised July 1977

(11)

(12) 36p.

(15) NOOO14-75-C-0621,
NSF-MPS-73-08534

403 426

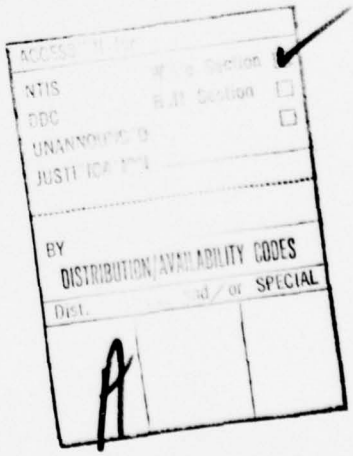
This report was prepared as part of the activities of the Management Sciences Research Group, Carnegie-Mellon University, under Grant MPS73-08534 A02 of the National Science Foundation and Contract N00014-75-C-0621 NR 047-048 with the U.S. Office of Naval Research. Reproduction in whole or in part is permitted for any purpose of the U.S. Government.

Management Science Research Group
Graduate School of Industrial Administration
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

*Currently with the Graduate School of Management, Northwestern University, Evanston, Illinois.

Abstract

We describe an algorithm for the 0-1 knapsack problem (KP), which relies mainly on three new ideas. The first one is to focus on what we call the core of the problem, namely a knapsack problem equivalent to (KP), defined on a particular subset of the variables. The size of this core is usually a small fraction of the full problem size, and does not seem to increase with the latter. While the core cannot be identified without solving (KP), a satisfactory approximation can be found by solving (LKP), the associated linear program. The second new ingredient is a binary-search-type procedure for solving (LKP) which, unlike earlier methods, does not require any ordering of the variables. The computational effort involved in this procedure is linear in the number of variables. Finally, the third new feature is a simple-minded heuristic whose accuracy under certain conditions grows exponentially with the problem size. Computational experience wit[illegible]gorithm based on the above ideas, on 200 randomly generated test problems with 1,000-10,000 variables and with coefficients ranging from between 10-100 to between 10-10,000, indicates that for such problems the computational effort grows linearly with the number of variables and logarithmically with the range of coefficients. Total time for the 200 problems was 160 UNIVAC 1108 seconds, and the maximum time for any single problem was 3 seconds.



SOLVING LARGE ZERO-ONE KNAPSACK PROBLEMS

by

Egon Balas and Eitan Zemel

1. Introduction

We consider the 0-1 knapsack problem

$$\text{(KP)} \qquad \max \{cx \mid ax \leq a_0,\ x_j = 0 \text{ or } 1,\ j \in N\} \ ,$$

where $c = (c_j)$ and $a = (a_j)$ are positive n-vectors with integer components, a_0 is a positive integer, and $N = \{1,\ldots,n\}$. We will also refer to the linear program associated with (KP),

$$\text{(LKP)} \qquad \max \{cx \mid ax \leq a_0,\ 0 \leq x \leq e\} \ ,$$

where $e = (1,\ldots,1)$ has n components. Our approach can be extended in a straightforward manner to the general bounded-variables (as opposed to 0-1) knapsack problem.

Several fast algorithms are available for the solution of (KP) (see, for instance, [4], [10], [13]), based on first solving (LKP) and generating an integer solution to define upper and lower bounds, then using these bounds with some logical tests to fix as many variables as possible, and finally solving the knapsack problem in the remaining variables by some specialized version of implicit enumeration.

The starting point of each of these algorithms is the ordering of the variables according to decreasing cost-to-weight (c_i/a_i) ratios, which is the basis of the method used for solving (LKP). This preprocessing of the variables, though often done by a separate sorting routine and therefore (?) not included in the total times reported, usually constitutes the lion's share of the computational effort required to solve (KP).

We propose an approach to the 0-1 knapsack problem which, while using in its final stage some of the above mentioned techniques, relies mainly on three new ideas. The first one is to focus on what we call the core problem, i.e., the subproblem in those variables, whose cost/weight ratio falls between the maximum and the minimum c_i/a_i ratio for which x_i has a different value in an optimal solution to (KP) from that in an optimal solution to (LKP) (see section 2). The size of this core is usually a surprisingly small fraction of the full problem size, and does not increase with the latter. While precise identification of the core would require the solving of (KP), a satisfactory approximation can be found by solving (LKP).

The second new ingredient is a binary-search-type method for solving (LKP) without ordering the variables (see section 3). As a byproduct, this procedure also yields a convenient approximation (IP) to the core problem. The classical procedure for solving (LKP) is based on ordering the variables according to their cost/weight ratios, an operation which by itself is of complexity $O(n\log_2 n)$. The computational complexity of our procedure is $O(n)$.

The third idea is to make use of the fact that, if the cost-to-weight ratios of the approximate core problem are close enough to each other (a condition that (IP) tends to satisfy by construction), there exists a simple heuristic which finds an optimal solution to (IP) with a probability that grows exponentially with the size of (IP) (see section 4).

Procedures based on the above three ideas usually produce an integer solution which is either optimal, or provides a sufficiently good lower bound to make it possible to fix the bulk of the variables outside the approximate core problem. This is done by a slightly improved version of some logical tests from the literature, and is followed by implicit enumeration on the remaining variables (see section 5).

Computational experience on 200 randomly generated test problems with 1000 - 10000 variables, and with coefficients ranging from 10 - 10000, discussed in section 6, indicates that for such problems the computational effort (about 160 UNIVAC 1108 seconds for the 200 problems, the maximum time for any single problem being 3 seconds) grows linearly with the number of variables, and logarithmically with the range of coefficients. This makes it possible to solve randomly generated 0-1 knapsack problems of virtually any size, by a computational effort which (for large enough problems) is smaller than the effort required to order the variables. Problems whose cost-to-weight ratios are all within a small interval (or are all equal), which have been found difficult for most other methods, are easily handled by our approach.

On the other hand, there is a class of 0-1 knapsack problems that is hard for this method (as well as for others). In section 7 we identify this class of problems, and outline a cutting plane method which, while unable to compete with the above approach on randomly generated problems, is often superior on problems in the "hard" category.

2. The Core Problem and Its Approximations

In order to define the core problem mentioned in section 1, we will assume N to be ordered so that

$$\frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \cdots \geq \frac{c_n}{a_n} .$$

For an optimal solution to (KP), define

$$j_* = \min \{ j \in N \mid \hat{x}_j = 0 \}$$

$$j_{**} = \max \{ j \in N \mid \hat{x}_j = 1 \} ,$$

and

$$j_1 = \min \{ j_*, j_{**} \} , \qquad j_2 = \max \{ j_*, j_{**} \} .$$

Note that $j_1 = j_*$ except for the case when $\hat{x}$, with its components ordered as specified above, is of the form $(1,\ldots,1,0,\ldots,0)$.

If (KP) has more than one optimal solution, let $\hat{x}$ be one which minimizes the size of the interval $C = [j_1, j_2]$. We will call C the core of N, and the knapsack problem

$$(CP) \qquad \max \{ \sum_{j \in C} c_j x_j \mid \sum_{j \in C} a_j x_j \leq a_0 - \sum_{j=1}^{j_1-1} a_j, \; x_j = 0 \text{ or } 1, \; j \in C \}$$

the core of (KP), or the core problem associated with (KP).

The core problem is easily seen to be equivalent to (KP) in the following sense.

Proposition 1. $\bar{x} \in R^{|C|}$ is optimal for (CP) if and only if $\hat{x}$ is optimal for (KP), where

$$\hat{x}_j = \begin{cases} \bar{x}_j & j \in C \\ 1 & j < j_1 \\ 0 & j > j_2 \end{cases} .$$

Proof. If $\tilde{x}$ is a solution to (CP) better than $\bar{x}$, then $\tilde{x}'$, defined relative to $\tilde{x}$ in the same way as $\hat{x}$ is defined relative to $\bar{x}$, is a solution to (KP) better than $\hat{x}$, a contradiction. Conversely, if (KP) has a solution better than $\hat{x}$, its projection on the subspace corresponding to C is a solution to (CP) better than $\bar{x}$, again a contradiction. ||

Another essential property of the core is the following.

Proposition 2. (LKP) has an optimal solution $\bar{x}$ such that $\bar{x}_j = 0$, $\forall j < j_1$, and $\bar{x}_j = 1$, $\forall j > j_2$.

Proof. Let $\hat{x}$ be the solution to (KP) used to define j_1 and j_2. With N ordered according to decreasing c_i/a_i ratios, (LKP) has an optimal solution $\bar{x}$ of the form $\bar{x}_j = 1$, $j < f$, $0 \leq \bar{x}_f \leq 1$, $\bar{x}_j = 0$, $j > f$, for some $f \in N$. If

$\bar{x}_f = 0$, then $\hat{x}$ is also an optimal solution to (LKP) and the proposition holds. Now let $0 < \bar{x}_f < 1$. If $f < j$, then $c\hat{x} > c\bar{x}$, which contradicts the optimality of $\bar{x}$ for (LKP). If $f > j$, then $\tilde{x}$ defined by $\tilde{x}_j = 1$, $j = 1,\ldots,j_2$, $\tilde{x}_j = 0$, $j > j_2$ is a feasible solution to (KP), with $c\tilde{x} > c\hat{x}$, which contradicts the optimality of $\hat{x}$ for (KP). ||

For randomly generated 0-1 knapsack problems the size of the core is usually a small fraction of the size of the full problem, and does not seem to depend on the latter (25 is as typical a core size for 100-variable problems as it is for problems with 10,000 variables). Since the core, as defined above, cannot be identified without knowing the optimal solutions of (KP), an idea which suggests itself in a natural way is to try to approximate the core. Any interval $I \subset N$ (where N is ordered as above) containing the index of the fractional component of an optimal solution to (LKP) can be viewed as an approximation to the core. We will denote by (IP) the approximate core problem defined by $I = [i_1,i_2]$, i.e.,

$$\text{(IP)} \qquad \max \{ \sum_{i\in I} c_i x_i \mid \sum_{i\in I} a_i x_i \leq a_0 - \sum_{i=1}^{i_1-1} a_i ,\ x_i = 0 \text{ or } 1,\ i \in I \} \ .$$

The choice of I involves a tradeoff between the probability that $I \supseteq C$ (i.e., that the resulting (IP) is equivalent to (KP)), and the degree of difficulty of solving (IP).

At first glance, it would seem that approximating the core requires that we solve (LKP) by ordering N according to decreasing c_i/a_i ratios. In the next section we give a procedure which finds an optimal solution $\bar{x}$ to (LKP) without ordering N, and as a byproduct yields a reasonably sized (for randomly generated problems) interval $I = [i_1,i_2]$, $I \subset N$, containing the index of the fractional component of $\bar{x}$.

3. Solving (LKP) Without Sorting

It is well known that if N is ordered according to decreasing c_i/a_i ratios, then an optimal basic solution $\bar{x}$ to (LKP) is of the form

$$\bar{x}_j = \begin{cases} 1 & j < f \\ 0 & j > f \\ a_0 - \sum_{j=1}^{f-1} a_j & j = f \end{cases}$$

where f is the smallest $j \in N$ such that $\sum_{i=1}^{j} a_i \geq a_0$. The classical method of solving (LKP) is based on this observation. Since sorting n elements requires by itself <u>at least</u> $cn\log_2 n$ comparisons (c a constant) (see [1], Corollary to Theorem 3.4), solving (LKP) by the classical procedure requires at least $(\log_2 n)$ operations.

Since the ordering of N is not unique, neither is the index f. However, the ratio c_f/a_f is unique, and will henceforth be called the <u>critical</u> ratio. For any positive scalar λ, denote

$$S_1(\lambda) = \sum_{i \mid (c_i/a_i) > \lambda} a_i , \qquad S_2(\lambda) = S_1(\lambda) + \sum_{i \mid (c_i/a_i) = \lambda} a_i .$$

Then the critical ratio $\lambda^* = c_f/a_f$ is obviously the unique solution to the pair of inequalities

$$S_1(\lambda) < a_0 \leq S_2(\lambda) ,$$

and solving (LKP) essentially amounts to finding the critical ratio λ^*.

The algorithm below (Procedure 1) accomplishes this without sorting the variables. In its statement, the ordering of N is arbitrary. At each iteration, N_0, N_1 and N_F denote the index sets of the variables fixed at 0,

fixed at 1, and free, respectively. The sums $S_1(\lambda)$ and $S_2(\lambda)$ are always defined on the current set of free variables.

Procedure 1.

0. Initialize: set $N_0 = \emptyset$, $N_1 = \emptyset$, $N_F = N$, and go to 1.

1. Choose $\lambda \in \{c_i/a_i\}_{i \in N_F}$, partition N_F into

$$N^{>} = \{i \in N_F \mid (c_i/a_i) > \lambda\}$$

$$N^{<} = \{i \in N_F \mid (c_i/a_i) < \lambda\}$$

$$N^{=} = \{i \in N_F \mid (c_i/a_i) = \lambda\} \quad ,$$

and calculate $S_1(\lambda)$, $S_2(\lambda)$. Go to 2.

2. If $S_1(\lambda) < a_0 \leq S_2(\lambda)$, stop: $\lambda^* = \lambda$. An optimal solution $\bar{x}$ to (LKP) is obtained by setting

$$\bar{x}_j = \begin{cases} 1 & j \in N_1 \cup N^{>} \\ 0 & j \in N_0 \cup N^{<} \end{cases}$$

and then "filling the knapsack" with variables x_j, $j \in N^{=}$ (any, possibly including one at a fractional value).

If $S_1(\lambda) \geq a_0$, i.e., if λ is too small, set $N_0 \leftarrow N_0 \cup N^{<} \cup N^{=}$, and $N_F \leftarrow N^{>}$.

If $S_2(\lambda) < a_0$, i.e., if λ is too large, set $N_1 \leftarrow N_1 \cup N^{>} \cup N^{=}$, and $N_F \leftarrow N^{<}$.

Then go to 3.

3. If $N_F > \theta$ (where θ is a fixed threshold value), go to 1. Otherwise solve the linear program defined on N_F by the usual method (i.e., by ordering N_F) and extend the solution to (LKP) by setting $x_j = 1$, $j \in N_1$, $x_j = 0$, $j \in N_0$. This solution is optimal.

The threshold value θ in step 3 is meant to identify the problem size under which the usual solution method (based on sorting the variables) becomes preferable to the one consisting of steps 1-2. We have found $\theta = 25$ an adequate value.

As for the value of λ in step 1, the ideal choice would of course be the median of the ratios c_i/a_i, $i \in N_F$. Calculating the median involves, however, a certain computational effort. Instead, we have tried the following two rules: (a) choose λ to be the ratio c_i/a_i closest to $\frac{1}{2}\left(\max_{i \in N_F} \frac{c_i}{a_i} + \min_{i \in N_F} \frac{c_i}{a_i}\right)$; (b) choose λ to be the median of the first k ratios c_i/a_i encountered. Rule (b) with $k = 3$ turned out to be preferable to (a).

As a byproduct of solving (LKP), Procedure 1 yields a natural candidate for the approximate core problem (IP), namely the one defined by setting

$$I = \begin{cases} N_F & \text{if } |N_F| \leq \theta \\ N_F^{\theta} & \text{otherwise} \end{cases},$$

where N_F is the last set of free variables, ordered according to decreasing c_i/a_i ratios, while N_F^{θ} is an (ordered) subset of N_F such that $|N_F^{\theta}| = \theta$ and f, the index of the fractional component of $\bar{x}$, is contained in the middle third of N_F^{θ}.

The size of I is thus controlled by the parameter θ, which can be chosen equal to, or slightly greater than, the desired core size. However, the actual size of the last set N_F may be considerably smaller than θ. Also, while N_F is guaranteed to contain the fractional component of $\bar{x}$, this component may happen to be closer to one of the endpoints of the interval defined by N_F than to its center, a configuration which does not make for a good approximation to the core problem. When one of these situations arises,

Procedure 1 "enlarges" the interval defined by N_F in the desired direction, by adding to it the last set $N^{>} \cup N^{=}$ included into N_1 in step 2 (if the enlargement is to be made "to the left"), or the last set $N^{<} \cup N^{=}$ included into N_0 in step 2 (if one wants an enlargement "to the right"). These sets are again natural candidates for the enlargement of I, since their c_i/a_i ratios are closest to those in N_F.

Finally, Procedure 1 also produces a first integer solution $\tilde{x}$. Namely, if the optimal solution found for (LKP) is $\bar{x}$, Procedure 1 sets $\tilde{x}_j = 1$ if $\bar{x}_j = 1$, $\tilde{x}_f = 0$, then examines each remaining variable in turn and, if possible, sets it to 1.

Next we show that the computational effort involved in Procedure 1 is linear in n, the number of variables. To analyze the worst-case behavior of this algorithm, we specify the choice of λ in step 1 as that of the median of the ratios c_i/a_i, $i \in N_F$. Calculating the median of p numbers requires a computational effort linear in p; more precisely, it requires at most 3p comparisons [11].

The computational effort involved in Procedure 1 can then be described as follows.

Preprocessing: n divisions, to calculate the ratios c_i/a_i.

Step 1: at most $3|N_F|$ comparisons to calculate the median of the ratios c_i/a_i, $i \in N_F$, plus $|N_F|$ comparisons and $|N_F|$ additions to partition N_F and calculate $S_1(\lambda)$, $S_2(\lambda)$, i.e., a total of $5|N_F|$ operations.

Step 2: two comparisons, plus the transfer of 3 subsets (which involves changing 6 pointers), i.e., 8 operations.

If t is the number of iterations, steps 1 and 2 are executed t times.

If the algorithm ends at step 2, there are an additional $|N_F|$ comparisons and $|N_F|$ additions to be performed in order to assign values to the variables x_i, $i \in N_F$.

Step 3: this requires a number of at most $c = O(\theta \log_2 \theta)$ operations (sorting at most θ elements, plus assigning values to the corresponding variables). This number is independent of n.

Since $|N_F| = n$ at the start, and N_F is reduced by at least $\frac{1}{2}$ at each iteration, at the end of the procedure $|N_F| \leq \frac{1}{2^t} n$.

Let w_1 denote the time needed to perform a division, and w_2 the time needed for a comparison, addition or change of pointers. Then if W(n) denotes the maximum time needed to solve a problem in n variables by Procedure 1, we have

$$W(n) \leq w_1 n + w_2 \left[5n\left(1 + \frac{1}{2} + \ldots + \frac{1}{2^t}\right) + 8t + 2 \cdot \frac{1}{2^t} n + c \right]$$

$$\leq w_1 n + w_2 \left[5n\left(1 + \frac{1}{2} + \ldots + \frac{1}{2^{t+1}}\right) + 8t + c \right]$$

$$\leq (w_1 + 10w_2)n + 8w_2 \log_2 n + w_2 c.$$

Hence $W(n) = O(n)$.

A similar analysis of the <u>expected</u> time T(n) needed to solve a problem in n variables by Procedure 1, with λ chosen randomly in step 1, shows that T(n) is given by an expression of the same form as the above bound for W(n), with the coefficient of n slightly smaller than $w_1 + 10w_2$. Computational experience with Procedure 1 is to be found in section 6, where we compare its performance to that of two sorting routines from the literature. The comparison is unequivocally favorable to Procedure 1.

4. A Heuristic Whose Accuracy Improves Exponentially With Problem Size

Once the approximate core problem (IP) has been defined, we use a heuristic (Procedure 2 below) to find a "good" integer solution to (IP).

Step 3: this requires a number of at most $c = O(\theta \log_2 \theta)$ operations (sorting at most θ elements, plus assigning values to the corresponding variables). This number is independent of n.

Since $|N_F| = n$ at the start, and N_F is reduced by at least $\frac{1}{2}$ at each iteration, at the end of the procedure $|N_F| \leq \frac{1}{2^t} n$.

Let w_1 denote the time needed to perform a division, and w_2 the time needed for a comparison, addition or change of pointers. Then if $W(n)$ denotes the maximum time needed to solve a problem in n variables by Procedure 1, we have

$$W(n) \leq w_1 n + w_2 \left[5n\left(1 + \frac{1}{2} + \ldots + \frac{1}{2^t}\right) + 8t + 2 \cdot \frac{1}{2^t} n + c \right]$$

$$\leq w_1 n + w_2 \left[5n\left(1 + \frac{1}{2} + \ldots + \frac{1}{2^{t+1}}\right) + 8t + c \right]$$

$$\leq (w_1 + 10w_2)n + 8w_2 \log_2 n + w_2 c.$$

Hence $W(n) = O(n)$.

A similar analysis of the <u>expected</u> time $T(n)$ needed to solve a problem in n variables by Procedure 1, with λ chosen randomly in step 1, shows that $T(n)$ is given by an expression of the same form as the above bound for $W(n)$, with the coefficient of n slightly smaller than $w_1 + 10w_2$. Computational experience with Procedure 1 is to be found in section 6, where we compare its performance to that of two sorting routines from the literature. The comparison is unequivocally favorable to Procedure 1.

4. A Heuristic Whose Accuracy Improves Exponentially With Problem Size

Once the approximate core problem (IP) has been defined, we use a heuristic (Procedure 2 below) to find a "good" integer solution to (IP).

If I_1 was exhausted, stop; otherwise set $i \leftarrow i+1$ and go to the Iterative Step.

The reason for not setting to 1 in the first instance those x_j such that $u_i - a_k < MIN$, is that doing so would produce an integer solution which does not satisfy the knapsack constraint with equality. But the main rationale behind this simple-minded heuristic is that, by construction, (IP) has all of its ratios c_i/a_i within a small range around the ratio c_f/a_f; and thus an integer solution to (IP) which satisfies the knapsack constraint with equality is likely to be close to the optimum, and so is the associated solution to (KP).

To make this statement more precise, let $\bar{r} = \max_{i \in I} \frac{c_i}{a_i}$, $\underline{r} = \min_{i \in I} \frac{c_i}{a_i}$, and $u_* = \max_{i \in I_1} u_i$. Also, let v(KP) and v(LKP) be, as before, the value of (an optimal solution to) KP and LKP respectively, and let v_H be the value of the solution obtained by the above heuristic. Then, on the one hand,

$$v_H \leq v(KP) \leq v(LKP)$$

but on the other,

$$v_H \geq v(LKP) - (\bar{r} - \underline{r})u_*$$

since u_* is an upper bound on the sum of coefficients a_i such that $\bar{x}_i = 1$, $\hat{x}_i = 0$, and $\bar{r} - \underline{r}$ is an upper bound on the change in the objective function value caused by a unit change in that sum. From the last inequality,

$$v(LKP) - v_H \leq (\bar{r} - \underline{r})u_* ;$$

hence if

$$\bar{r} - \underline{r} < \frac{1}{u_*} ,$$

then $v(LKP) - v_H < 1$, i.e., any integer solution that satisfies the knapsack constraint with equality, is optimal.

Let $\bar{x}$ be the optimal solution to (LKP) found by Procedure 1, let $\bar{x}_f$ be its fractional component, and let the approximate core problem be

$$(IP) \qquad \max \{ \sum_{i \in I} c_i x_i \mid \sum_{i \in I} a_i x_i \leq \bar{a}_0 , \; x_i = 0 \text{ or } 1, \; i \in I \} ,$$

where

$$\bar{a}_0 = a_0 - \sum_{i=1}^{i_1 - 1} a_i ,$$

while $I = \{i_1, \ldots, i_t\}$ is the ordered set obtained at the end of Procedure 1, which contains the index f of the fractional variable. Let $MAX = \max_{i \in I} a_i$, $MIN = \min_{i \in I} a_i$, partition the ordered set I into

$$I_1 = \{i_1, \ldots, f-1\} \quad \text{and} \quad I_0 = \{f, \ldots, i_t\} ,$$

and for each $i \in I_1$, define $u_i = a_i + a_f \bar{x}_f$.

Since the knapsack inequality holds with equality for $\bar{x}$, u_i is the "unfilled capacity" created by setting x_f and x_i to 0.

Procedure 2. Set $\hat{x}_f = 0$, $i \leftarrow i_1$ and go to the

Iterative Step. Set $\hat{x}_i = 0$ and try to match the "unfilled capacity" u_i with coefficients a_k, $k \in I_0$. To do this, consider in turn each $k \in I_0$ and set

$$\hat{x}_k = 1, \; u_i \leftarrow u_i - a_k , \qquad \text{if } u_i = a_k, \text{ or } u_i - a_k \geq MIN$$

$$\hat{x}_k = 0, \; u_i \leftarrow u_i , \qquad \text{if } u_i - a_k < MIN .$$

If as a result $u_i = 0$ is obtained, complete the solution $\hat{x}$ by setting $\hat{x}_i = \bar{x}_i$ for all components whose value has not yet been assigned, compare $\hat{x}$ with the best integer solution at hand, and store the better of the two.

If $u_i = 0$ is not obtained, set $\hat{x}_k = 1$ for the first $k \in I_0$ such that $u_i - a_k < MIN$; then complete $\hat{x}$ and proceed as above.

If I_1 was exhausted, stop; otherwise set $i \leftarrow i+1$ and go to the Iterative Step.

The reason for not setting to 1 in the first instance those x_j such that $u_i - a_k < MIN$, is that doing so would produce an integer solution which does not satisfy the knapsack constraint with equality. But the main rationale behind this simple-minded heuristic is that, by construction, (IP) has all of its ratios c_i/a_i within a small range around the ratio c_f/a_f; and thus an integer solution to (IP) which satisfies the knapsack constraint with equality is likely to be close to the optimum, and so is the associated solution to (KP).

To make this statement more precise, let $\bar{r} = \max_{i \in I} \frac{c_i}{a_i}$, $\underline{r} = \min_{i \in I} \frac{c_i}{a_i}$, and $u_* = \max_{i \in I_1} u_i$. Also, let v(KP) and v(LKP) be, as before, the value of (an optimal solution to) KP and LKP respectively, and let v_H be the value of the solution obtained by the above heuristic. Then, on the one hand,

$$v_H \leq v(KP) \leq v(LKP)$$

but on the other,

$$v_H \geq v(LKP) - (\bar{r} - \underline{r})u_*$$

since u_* is an upper bound on the sum of coefficients a_i such that $\bar{x}_i = 1$, $\hat{x}_i = 0$, and $\bar{r} - \underline{r}$ is an upper bound on the change in the objective function value caused by a unit change in that sum. From the last inequality,

$$v(LKP) - v_H \leq (\bar{r} - \underline{r})u_* ;$$

hence if

$$\bar{r} - \underline{r} < \frac{1}{u_*} ,$$

then $v(LKP) - v_H < 1$, i.e., any integer solution that satisfies the knapsack constraint with equality, is optimal.

Suppose now that this condition is satisfied, i.e., the difference between the best and worst c_i/a_i ratio in the approximate core problem is less than $1/u_*$. We wish to examine the probability that Procedure 2 fails to find an optimal solution to (IP), i.e., to find an integer solution which "fills the knapsack".

Procedure 2 systematically tries to match the coefficients a_k, $k \in I_0$, against the values $u_i = a_i + a_f\bar{x}_f$, $i \in I_1$. If the coefficients $k \in I_0$ are uniformly distributed over the set of integers between MIN and MAX, and are independent of each other, the number of possible distinct values for them is $V_0 = MAX - MIN + 1$. Let V_1 denote the number of distinct values u_i, $i \in I_1$, such that $MIN \leq u_i \leq MAX$. (By definition of the u_i, some of them may be outside this interval.) Thus we have a set of V_1 distinct values u_i from [MIN, MAX], and a set of $|I_0|$ (not necessarily distinct) values a_k, uniformly distributed over that same interval. The probability that any particular value a_k, $k \in I_0$, does not match any of the V_1 distinct values u_i is $1 - \frac{V_1}{V_0}$. The probability that none of the $|I_0|$ values a_k, $k \in I_0$, matches any of the values u_i is then $\left(1 - \frac{V_1}{V_0}\right)^{|I_0|}$. This is actually an overestimate of the probability of the failure of Procedure 2 to "fill the knapsack," since it ignores the possibility of each u_i, $i \in I_1$, being matched by the sum of several a_k, $k \in I_0$, rather than by a single a_k.

The remarkable fact about this simple-minded heuristic is that, if the c_i/a_i ratios of (IP) are sufficiently close to each other, the probability that it fails to find an optimal solution to (IP) decreases exponentially with $|I_0|$, which is more or less the same as exponentially with the size of (IP). Values of the above expression for certain values of its parameters are given in Table 1.

Table 1. Values of $\left(1 - \frac{v_1}{v_0}\right)^{|I_0|}$.

$v_0 = 50$

v_1 \ $\lvert I_0 \rvert$	5	10	15	20
5	.59	.35	.2	.12
10	.32	.11	.03	.01
15	.16	.03	.004	.0008

$v_0 = 500$

v_1 \ $\lvert I_0 \rvert$	10	30	50	70
10	.82	.54	.36	.24
30	.54	.15	.04	.01
50	.35	.042	.005	.0006

$v_0 = 100$

v_1 \ $\lvert I_0 \rvert$	10	20	30
10	.35	.12	.04
15	.20	.03	.007
20	.11	.01	.001

$v_0 = 1{,}000$

v_1 \ $\lvert I_0 \rvert$	25	50	75	100
25	.53	.28	.15	.08
50	.28	.08	.02	.006
75	.14	.02	.003	.0004

Thus, if the above heuristic is applied to a knapsack problem with, say, 200 variables, and with coefficients randomly drawn from the interval [10,1000], the probability of finding an optimal solution is about .9996; and the larger the problem (for the given coefficient interval), the higher this probability.

A slightly improved version of the heuristic described above makes use of the following dominance relation. If $i,j \in N$ are such that

$$c_i \geq c_j \quad , \qquad a_i \leq a_j \quad ,$$

we say that x_i <u>dominates</u> x_j. If at least one of the two inequalities holds strictly, we say that x_i <u>strictly</u> dominates x_j. Clearly, if x_i dominates x_j, then (KP) has an optimal solution in which $x_i \geq x_j$. As a result, if x_i is set to 0, then x_j can also be set to 0. Incorporating this feature in Procedure 2 improves the quality of the solutions generated, though for problems with a large coefficient range the effect is not significant. On the other hand, for problems with many equal c_i/a_i ratios, using the dominance relation in the above fashion may render the heuristic ineffective, by drastically reducing the number of coefficients a_k, $k \in I_0$, that one tries to match against each u_i, $i \in I_1$. This effect can be avoided by fixing variables only on the basis of strict (rather than simple) dominance.

5. Logical Tests and Implicit Enumeration

The sequence of Procedures 1 and 2 produces an integer solution $\hat{x}$ whose quality for randomly generated problems is usually very good. If the range of the problem coefficients is small (say, below 100), then the value v_H of the integer solution $\hat{x}$ found by the heuristic of section 4 often differs by less than 1 from $v(LPK) = c\bar{x}$, in which case of course $\hat{x}$ is optimal. If $v(LPK) - v_H > 1$, we apply slightly modified versions of some earlier procedures from the literature.

First, we use some simple logical tests (Procedure 3) to fix as many of the out-of-core variables as possible (for the in-core variables, these tests are applied simultaneously with the heuristic of section 4). The first of these tests (proposed by Dembo and Hammer [6]) sets permanently $x_j = \bar{x}_j$ if

the reduced cost of x_j matches (or exceeds) the above gap; i.e., if

$$|c_j - a_j(c_f/a_f)| \geq v(LKP) - v_H$$

(where $\bar{x}_f$ is, as before, the fractional component of $\bar{x}$). The second test (proposed by Ingargiola and Korsh [7]) sets tentatively, for each j in turn, $x_j = 1 - \bar{x}_j$, then solves the linear program in the remaining variables, and if the value of the solution equals or exceeds v_H, sets permanently $x_j = \bar{x}_j$. "Solving" the linear programs involved in this test amounts to the following. Let $I = [i_1, i_2]$ be the interval defining the approximate core problem (IP), and for $i < i_1$, denote by $(LKP)_{x_i=0}$ the linear program obtained from (LKP) by tentatively setting $x_i = 1 - \bar{x}_i = 0$. Then we set $x_j = \bar{x}_j$, $\forall\ j < f$, $j \neq i$, and $\forall\ j > i_2$, and then "fill the knapsack" by assigning maximal values to the variables x_j, $f \leq j \leq i_2$, taken in order of increasing j. Whenever this is possible, the resulting solution is obviously optimal for $(LKP)_{x_i=0}$. When the knapsack cannot be "filled" in this way, we approximate from above the optimum of $(LKP)_{x_i=0}$ by introducing a fictitious variable x_α whose weight-coefficient a_α "fills the knapsack" and whose cost coefficient satisfies $c_\alpha/a_\alpha = c_{i_2}/a_{i_2}$ (where i_2 is the last index of I). Such an overestimate of the optimum of $(LKP)_{x_i=0}$ clearly preserves the validity of the above described logical test. A perfectly analogous procedure is used when one tentatively sets $x_i = 1 - \bar{x}_i = 1$ for some $i > i_2$.

The reason for applying the above two tests in this particular order is that the second test is more powerful, but also considerably more expensive computationally, than the first one. It makes therefore sense to apply the second test to the reduced problem remaining after the first test has been used to fix an (often significant) number of variables.

The power of the second test can be enhanced at a low cost by using the dominance concept defined in section 4. Thus, whenever x_i fails the test and is therefore (permanently) set to $1 - \bar{x}_i$, we also set to $1 - \bar{x}_i$ all the variables dominated by x_i (if $1 - \bar{x}_i = 0$), or all the variables dominating x_i (if $1 - \bar{x}_i = 1$). The reason why this strengthens the procedure is that, contrary to intuition, it is possible for a variable x_i to fail the test, and for a variable x_j dominating x_i or dominated by x_i, to pass it.

Procedure 3 consisting of the above tests terminates by introducing all the variables that were not fixed into the approximate core problem (IP). If this redefined problem (IP) has more than s variables (s a parameter, say s = 50), then the heuristic (Procedure 2) is again applied to it, followed by the logical tests (Procedure 3). Otherwise we apply the final part of the algorithm, which is Procedure 4.

Procedure 4 is a virtually unchanged version of Zoltners' implicit enumeration procedure [13], called by its author a direct descent algorithm. (Another name, coined by Bradley [5], is fixed order enumeration.) The variables are ordered according to decreasing c_i/a_i ratios. At each node of the search tree, the free variables are scanned in the above order, and all those that "fit into the knapsack" are set to 1, while the others are set to 0. As a result, both forward steps and backtracks usually involve long chains of variables, i.e., many levels of the tree (a sequence of variables fixed at 0 can only be "freed" by freeing the variable fixed at 1 which precedes the sequence). For details, see [13].

The entire algorithm described in the last three sections is summarized in the flowchart of Fig. 1.

The algorithm discussed here is highly efficient on randomly generated problems (we discuss our computational experience in the next section). An interesting feature of our approach, however, lies in the fact that it is equally

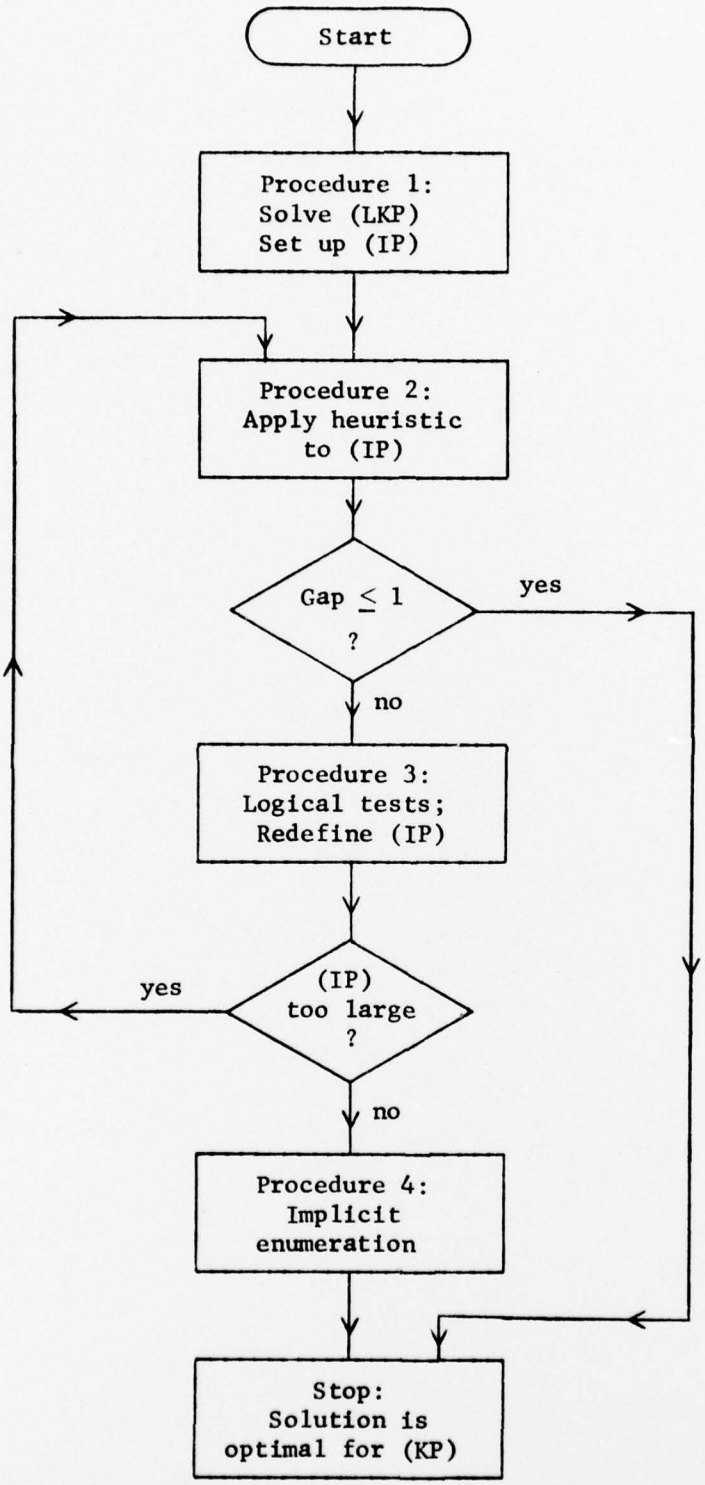

Start
Procedure 1:
Solve (LKP)
Set up (IP)
Procedure 2:
Apply heuristic
to (IP)
Gap ≤ 1
?
yes
no
Procedure 3:
Logical tests;
Redefine (IP)
(IP)
too large
?
yes
no
Procedure 4:
Implicit
enumeration
Stop:
Solution is
optimal for (KP)

Fig. 1

well (if not better) suited for problems whose c_i/a_i ratios are very close to each other. A recent survey of the 0-1 knapsack problem by S. Martello and P. Toth [9] has found that such problems tend to be very hard even for the most efficient implicit enumeration procedures. In particular, these procedures tend to break down in the case of the value-independent knapsack problem (i.e., the problem with c_i/a_i = const., $\forall i \in N$), mainly because, all the reduced costs being equal to 0, no variable can be fixed by the logical tests.

As to our algorithm, Procedures 1 and 2 are considerably more efficient on (randomly generated) value-independent knapsack problems, than on problems with randomly distributed c_i/w_i ratios. Procedure 1 solves (LKP) in one step, since the first λ chosen is critical. Procedure 2, if it "fills the knapsack", finds an optimal solution which is also instantly proved optimal, since $v_H = v(LPK)$.

On the other hand, if the heuristic does not "fill the knapsack", then applying the logical tests of Procedure 3 is useless, since in view of the 0 reduced costs these tests must fail. However, since all the c_i/a_i ratios are equal, the probability of Procedure 2 failing to find an optimal solution decreases exponentially with the size of (IP). Therefore, rather than applying Procedures 3 and 4, the thing to do in this case is to increase (say, double) the size of (IP) and apply again Procedure 2. This can be repeated, if necessary, but the data of Table 1 show how low the probability is of a need for repetition.

6. Computational Experience

The algorithm described in the previous sections was coded in FORTRAN IV. Efficient handling of the subsets of N and easy access to each of those subsets was accomplished by using a linked list (circular list)

data structure of the kind used by Barr and Ross [4] (and also Zoltners[13]). The code was tested on 200 problems with randomly generated objective function and constraint coefficients, and with $a_0 = \frac{1}{2} \sum_{i \in N} a_i$. To examine the behavior of the algorithm as a function of both problem size (number of variables) and coefficient range, we generated 20 classes of problems (10 in each class), with 1000, 2500, 5000, 7500 and 10000 variables respectively, and with coefficients randomly drawn from the intervals [10,50], [10,100], [10,1000] and [10,10000] respectively. The results are summarized in Table 2, each of whose entries gives the average computing time for 10 problems of the size and characteristics shown by the respective row and column heading, with the exception of the last two columns, where each entry gives the maximum and minimum time, respectively, used for any one problem with the number of variables indicated by the row heading, irrespective of the coefficient range. Times are reported in UNIVAC 1108 milliseconds, and include everything except the time used to generate the data and print out the results.

The times reported in Table 1 were obtained with a version of the code which conserves memory space (a scarce commodity on C-MU's UNIVAC 1108) by assigning only 4 integer memory units to each variable x_i (c_i, a_i, plus forward and backward pointers). In particular, the c_i/a_i ratios are not stored in this version, but recalculated each time they are needed. For some smaller problems (1000 and 5000 variables) we have also tested a version of the code in which these ratios are calculated and stored when the data are generated, and retrieved as they are needed. The execution time for Procedure 1 (which is between 40-60% of the total time) is then reduced to about 35% of the time needed in the version reported in Table 2 (see last column of Table 3).

Table 2. UNIVAC 1108 milliseconds used for solving (KP)

(each entry is the average time for 10 problems)

No. of variables \ Range of Coefficients	10-50	10-100	10-1000	10-10000	MAX	MIN
1000	149	170	234	250	403	49
2500	256	317	490	507	831	190
5000	561	567	1007	944	1968	407
7500	991	918	1424	1531	2179	604
10000	1153	992	2032	2078	3066	810

A striking feature of the results reported in Table 2 is the fact that the computational effort seems to grow linearly with the number of variables, and logarithmically (less than linearly) with the range of the problem coefficients. Even more impressive is the small difference between the average and the maximum times required to solve the 40 problems with 10000 variables shown in the last row of the tableau. While the average time for the two sets of 10 problems with coefficients in the 10-1000 and in the 10-10000 range respectively, and with 10000 variables each, is about 2 seconds, the maximum time for any of the 40 problems is about 3 seconds!

To assess the efficiency of the various components of our algorithm separately, we generated some additional data. Table 3 below compares the time needed by Procedure 1 to solve (LKP), generate a first integer solution and define the approximate core problem (IP), with the time needed by two sorting routines just to order the variables according to decreasing c_i/a_i ratios.

The first of these sorting routines, SORT, is Singleton's improved version of Quicksort [12]. It orders an array of integers according to decreasing values. The time used by SORT to accomplish this is in fact an

underestimate of the time needed for preprocessing the data in order to solve (LKP) in the traditional way, since SORT handles at each step only one entry for each $i \in N$, whereas in the case of an (LKP) one needs to handle at least 4 entries for each $i \in N$: c_i, a_i, c_i/a_i, as well as some indication of the variable's original identity. If the ratios c_i/a_i are calculated at each comparison, the time needed for sorting is even larger.

The second sorting routine, SORT 6, is the one used by Nauss in his algorithm for the 0-1 knapsack problem [10], and which performs all the necessary preprocessing of the data (it calculates and stores the c_i/a_i ratios, retrieving them as needed). Both sorting routines were used as originally coded by their authors.

Since Procedure 1 accomplishes more than SORT and is therefore, strictly speaking, only comparable to SORT 6, we also ran the problems with 1000 and 5000 variables respectively,with a version of Procedure 1 (called 1*) which solves (LKP) by retrieving the pre-stored c_i/a_i ratios as needed.

Each entry in Table 3 represents the average UNIVAC 1108 time for 10 problems with coefficients randomly drawn from the interval [10,10000]. Times are in milliseconds and include everything except the time used to generate the data and print out the results.

Table 3 speaks for itself. Furthermore, comparing its data with those of Table 2 shows that the algorithm discussed in this paper solves randomly generated large 0-1 knapsack problems in less time than it takes to order the variables according to decreasing c_i/a_i ratios.

The central idea of our approach is to concentrate on the core of the knapsack problem, approximated by (IP). The effectiveness of this is revealed dramatically by Table 4, which shows the results of constructing the approximate core problem (IP) by Procedure 1 and applying to it the (heuristic) Procedure 2,

Table 3. UNIVAC 1108 milliseconds used for sorting the variables (col. 1 and 2) versus solving (LKP) (col. 3 and 4)

(each entry is the average time for 10 problems)

No. of variables \ Code	1 SORT	2 SORT 6	3 Procedure 1	4 Procedure 1*
1000	148	536	114	37
2500	418	1751	270	—
5000	878	4163	473	166
7500	1377	6206	859	—
10000	1855	9645	928	—

as measured by the quality of the integer solution found and the number of variables left. This table refers to the same set of 200 randomly generated problems as Table 2. Again, each entry gives the average for 10 problems. The first three columns refer to the gap between v(LKP), the value of the linear programming optimum, on the one hand, and the value of the first integer solution, produced by Procedure 1 (col. 1), the value v_H of the integer solution constructed by the (heuristic) Procedure 2 (col. 2), and the value of the integer optimum (col. 3), on the other. Column 4 gives the relative error of the integer solution found by Procedure 2, expressed as the ratio $(v_H - v(KP)/v(KP)$. Column 5 shows the number of variables left in the core after one application of Procedure 2, which combines the heuristic for "solving" (IP) with some logical tests for fixing variables of (IP). Finally, column 6 shows the number of out-of-core variables left after Procedure 3, which uses logical tests based on the relatively narrow gap established by Procedures 1 and 2, to fix as many variables as possible.

Table 4. Effect of "solving" the approximate core problem (IP)

No. of Variables and Range of Coefficients	1 Gap after Procedure 1	2 Gap after Procedure 2	3 Final gap	4 Relative error	5 No. of variables in core after Procedure 2	6 No. of variables out of core after Procedure 3
N = 1000						
10-50	4.0	0.1	0.0	2.2×10^{-5}	1.4	0.0
10-100	5.4	1.6	0.9	2.4×10^{-5}	17.5	2.7
10-1000	59.1	12.5	9.7	1.1×10^{-5}	20.3	4.3
10-10000	527.4	159.0	106.9	2.1×10^{-5}	25.6	2.3
N = 5000						
10-50	3.5	0.1	0.0	1.3×10^{-6}	3.4	7.4
10-100	8.8	0.8	0.0	5.8×10^{-6}	17.9	29.2
10-1000	48.9	6.8	5.7	8.7×10^{-7}	24.8	13.7
10-10000	540.3	62.5	47.3	9.9×10^{-7}	43.0	19.2
N = 10000						
10-50	3.5	0.0	0.0	0.0	0.0	0.0
10-100	5.9	0.0	0.0	0.0	0.0	0.0
10-1000	59.3	4.4	3.3	4.3×10^{-7}	36.1	32.9
10-10000	463.3	37.5	31.5	2.3×10^{-7}	33.1	41.1

Table 4 shows that, given a large (KP), solving (LKP) (without sorting the variables) and then applying the heuristic to the small approximate core problem (20-50 variables) generated in the process, produces an integer solution within 10^{-5} — 10^{-7} of the optimum, and makes it possible to fix over 99% of the variables. The problem whose variables have to be sorted in order to apply to it the implicit enumeration algorithm of Procedure 4 thus has typically 30-70 variables.

It is instructive to observe the effect of problem size and coefficient range on the gaps listed in columns 1-3. Each of the 3 gaps seems to grow linearly with the coefficient range. However, while the first gap is almost unaffected by problem size (number of variables), the other two gaps tend to decrease with problem size. For the third gap, i.e., that between v(LPK) and the value of the integer optimum, this is due to the fact that for a given coefficient range, an increase in the number of variables tends to increase the number of c_i/a_i ratios which are close to the critical ratio, and thereby to increase the chances for the existence of an integer solution whose value is "close" to v(LKP). In other words, the combinatorial nature of the problem is mitigated by an increase in the number of variables. Similarly, in the case of the second gap, i.e., the one between v(LKP) and v_H, an increase in the number of variables (for a given coefficient range) tends to reduce the range of the ratios c_i/a_i spanned by the approximate core problem (IP), which in turn tends to increase the accuracy of the heuristic applied to (IP).

7. Hard Problems

The ease with which very large randomly generated 0-1 knapsack problems (including value-independent problems) can be solved, even when their coefficients span a sizeable range, is somewhat surprising in view of the fact,

established by Karp [8], that a polynomially bounded algorithm for this problem can only exist if one exists for the general integer program. This underscores the fact, already illuminated by the similar contrast between worst-case theoretical bounds and statistical behavior in the case of the simplex method, that often there exists little if any correlation between the worst case behavior and the expected behavior of algorithms. In particular, in the case of the 0-1 knapsack problem, the worst case behavior seems to be determined by a certain special structure which, in randomly generated problems, arises with a very low probability.

Since we have found that those 0-1 knapsack problems usually considered hard (and hard they are, for methods based on logical tests and implicit enumeration), i.e., problems whose c_i/a_i ratios are close to each other, are not hard at all for our approach, we set out to explore what problems are hard to handle by our method. The best measure of the degree of difficulty of a 0-1 problem that we could find was the relative size of the gap between the value of the linear programming optimum and that of the integer optimum, more precisely the ratio between this gap, $v(LKP) - v(KP)$, and the average size of the reduced costs $|c_j - a_j\lambda^*|$, $j \in N$ (here, as before, $\lambda^* = c_f/a_f$ is the critical cost/weight ratio). If we approximate the average reduced cost by 1/2 times the largest reduced cost, then this measure of the degree of difficulty becomes

$$\Delta = \frac{v(LKP) - v(KP)}{\frac{1}{2} \max_{i \in N} |c_i - a_i\lambda^*|} .$$

Knapsack problems tend to have a large Δ if there are relatively few integer solutions which satisfy the constraint with equality, and the optimum is not among them. We generated a set of 48 problems with this property, and found that only about 1/4 of them could be solved easily by our algorithm.

On the other hand, we have tried on this set of problems a cutting plane approach, whose efficiency on randomly generated problems comes nowhere near that of enumerative procedures, not to mention the algorithm discussed in this paper. Not entirely unexpectedly, the cutting plane approach turned out to be relatively efficient on the "hard" problems with large Δ.

The cutting planes used were those proposed in Theorem 2 of [2]. These inequalities are computationally cheap and in most cases (statistically, in more than 3/4 of the cases) are facets of the knapsack polytope, i.e., of the convex hull of feasible 0-1 points. When they are not facets, their coefficients are known [3] to differ by at most 1 from the corresponding coefficient of a facet (with the same right hand side).

The algorithm we used is as follows. First, Procedures 1, 2 and 3 of the previous sections are used to solve (LKP), find a good integer solution to (IP), and fix as many variables of (KP) as possible. If the problem in the remaining variables is sufficiently small, we solve it by Procedure 4 above. Otherwise, we generate an inequality which cuts off $\bar{x}$, the linear programming optimum, and solve the 2-constraint linear program (with the knapsack-inequality and the cut). The knapsack inequality is then replaced by a combination of the two constraints (with the optimal dual multipliers used as weights), which produces a new knapsack problem, with a smaller value of the linear program and hence with a smaller gap. We then apply Procedures 2 and 3 to this new knapsack problem, in which the smaller gap often leads to the fixing of new variables. The above sequence could be iterated for as long as a decrease in the gap can be obtained; but we stopped after the first cut, since the purpose of this preliminary experiment was to get a feel for the merits of the cutting plane approach in general.

We have tested this approach on 48 problems of the type described above, with 1000 variables each, and with coefficients in the range 1-100. Seven of the 48 problems were solved (optimality was proved) by Procedures 1 and 2. Another 5 problems were reduced by Procedures 1-2-3 to a sufficiently small size to be easily solved by Procedure 4. This makes for a total of 12 problems solved by the algorithm of sections 3-5. Of the remaining 36 problems, whose average Δ was .076, 18 were solved by the cut (optimality was established due to the tighter bound obtained by solving the 2-constraint linear program), while 7 were reduced to a size which made it easy to solve them by Procedure 4. In all these 25 problems the optimal solution was generated by the heuristic (Procedure 2), and the role of the cut was to make it possible to prove optimality. For the remaining 11 problems, the cut had practically no effect.

References

[1] A.V. Aho, J.E. Hopcroft, J.D. Ullman, The Design and Analysis of Computer Algorithms. Addison-Wesley, 1976.

[2] E. Balas, "Facets of the Knapsack Polytope." Mathematical Programming, 8, 1975, p. 146-164.

[3] E. Balas and E. Zemel, "Facets of the Knapsack Polytope from Minimal Covers," Management Science Research Report No. 352, Carnegie-Mellon University, December 1974. To appear in the SIAM Journal on Applied Mathematics, 1977.

[4] R.S. Barr and G.T. Ross, "A Linked List Data Structure for a Binary Knapsack Algorithm," Research Report CCS 232, Center for Cybernetic Studies, University of Texas, August 1975.

[5] G.H. Bradley, "Fixed Order Enumeration Methods." Paper presented at the ORSA/TIMS Meeting in Las Vegas, November 17-19, 1975.

[6] R.S. Dembo and P.L. Hammer, "A Reduction Algorithm for Knapsack Problems," Research Report CORR 75-6, Department of Combinatorics and Optimization, University of Waterloo. March 1975.

[7] G.P. Ingargiola and J.F. Korsh, "Reduction Algorithm for Zero-One Single Knapsack Problems," Management Science 20, 4 (1973) 460-463.

[8] R. Karp, "Reducibility Among Combinatorial Problems," in Complexity of Computer Computations, R.E. Miller and J.W. Thatcher (editors), Plenum Press, N.Y., 1972, p. 85-104.

[9] S. Martello and P. Toth, "The 0-1 Knapsack Problem." Paper presented at the Summer School on Combinatorial Optimization, SOGESTA, Urbino, May 30-June 11, 1977.

[10] R.M. Nauss, "An Efficient Algorithm for the 0-1 Knapsack Problem," University of California, January 1975 (Revised September 1975).

[11] A. Schönhage, M. Paterson and N. Pippenger, "Finding the Median." Theory of Computation Report No. 6, Computer Science Department, University of Warwick, 1975.

[12] R.C. Singleton, "Algorithm 347, An Efficient Algorithm for Sorting with Minimal Storage [M1]," CACM, Vol. 12, No. 3, March 1969, pp. 185-186.

[13] A.A. Zoltners, "A Direct Descent Binary Knapsack Algorithm," University of Massachusetts, September 1975.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: Technical Report No. 408
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): Solving Large Zero-One Knapsack Problems
5. TYPE OF REPORT & PERIOD COVERED: Technical Report July 1977
6. PERFORMING ORG. REPORT NUMBER: MSRR 408

7. AUTHOR(s): Egon Balas, Eitan Zemel
8. CONTRACT OR GRANT NUMBER(s): N00014-75-C-0621, MPS73-08534 A02 NSF

9. PERFORMING ORGANIZATION NAME AND ADDRESS: Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: NR 047-048

11. CONTROLLING OFFICE NAME AND ADDRESS: Personnel and Training Research Programs, Office of Naval Research (Code 434), Arlington, Virginia 22217
12. REPORT DATE: July 1977
13. NUMBER OF PAGES: 29

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):
15. SECURITY CLASS. (of this report): Unclassified
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

knapsack problem, integer programming, binary search, implicit enumeration

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

We describe an algorithm for the 0-1 knapsack problem (KP), which relies mainly on three new ideas. The first one is to focus on ~~what we call~~ the core of the problem, namely a knapsack problem equivalent to (KP), defined on a particular subset of the variables. The size of this core is usually a small fraction of the full problem size, and does not seem to increase with the latter. While the core cannot be identified without solving (KP), a satisfactory approximation can be found by solving (LKP), the associated

(continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

linear program. The second new ingredient is a binary-search-type procedure for solving (LKP) which, unlike earlier methods, does not require any ordering of the variables. The computational effort involved in this procedure is linear in the number of variables. Finally, the third new feature is a simple-minded heuristic whose accuracy under certain conditions grows exponentially with the problem size. Computational experience with an algorithm based on the above ideas, on 200 randomly generated test problems with 1,000-10,000 variables and with coefficients ranging from between 10-100 to be between 10-10,000, indicates that for such problems the computational effort grows linearly with the number of variables and logarithmically with the range of coefficients. Total time for the 200 problems was 160 UNIVAC 1108 seconds, and the maximum time for any single problem was 3 seconds.