A STABILITY CONCEPT FOR ZERO-ONE KNAPSACK PROBLEMS AND APPROXIMATION ALGORITHMS[1]

O. OGUZ

Department of Industrial Engineering, Bilkent University, Ankara, TURKIYE

M.J. MAGAZINE

Department of Management Science, University of Waterloo, Waterloo, Ont., N2L 3G1, CANADA

ABSTRACT

The concept of reducing the feasible range of decision variables or fixing the value of the variables is extended for the knapsack problem to include sets of variables. The ease of fixing these variables is measured by a stability index. The potential use of the concept is discussed in the context of approximation algorithms. Generalization to general zero-one problems is also considered.

Keywords: Knapsack problems, approximation algorithms.

RÉSUMÉ

Dans cet article, on étend au problème du sac alpin une méthode de fixation de variables, dans l'esprit des méthodes de réduction du domaine réalisable. La facilité avec laquelle on peut fixer des variables se mesure par un indice de stabilité. On discute de l'utlité de ce concept dans le cadre des alorithmes d'approximation. On considère aussi des généralisations aux problèmes généraux binaires.

Mots clés : problème du sac alpin; algorithmes d'approximation.

1. INTRODUCTION

Techniques aimed at fixing some of the variables of a discrete optimization problem, or reducing the range of those variables have been studied extensively and their specific application to knapsack problems is well known. The work in this area begins with Ingargiola and Korsh [10]. Then the works of Dembo and Hammer [4],Fayard and Plateau [5], [6], Nauss [16], Lauriere [11], Martello and Toth [15], and Balas and Zemel [1] explore fully the possibilities of applying the concept of reduction. These techniques are based on tightening the bounds for the variables through the use of available lower and upper bounds on the objective function. The bounds on the objective function are obtained by applying some relaxation or heuristic procedure. In the case of zero-one variables, reduction amounts to fixing some of the variables at either zero or one.

In this study we use the same concept to get additional information about some subsets of the variables of the problems rather than single variables. The term "stability", as formally defined in the next section will be the core of the analysis. Glover [7] uses the notion of "strongly determined and consistent variables" to convey similar ideas, but his definition is less formal and, in the final analysis, he restricts his attention to single variables.

2. DEFINITION OF STABILITY FOR THE ZERO-ONE KNAPSACK PROBLEM

The form of the knapsack problem we discuss is:

$$\text{(KP)} \quad \text{Max} \quad z = \sum_{i=1}^{n} c_i x_i \tag{1}$$

[1]Recd. June 90; Revd. Mar., Oct. 93 Acc. Mar. 94

$$\text{s.t.} \quad \sum_{i=1}^{n} a_i x_i \leq b,$$
$$x_i = 0 \text{ or } 1 \quad i \in N = \{1, 2, \ldots, n\}$$

We assume, without loss of generality, that $\sum_{i=1}^{n} a_i > b \geq a_i > 0$ and $c_i > 0$, $i = 1, \ldots, n$. The knapsack problem corresponds to trying to fill up a limited space with n indivisible items to yield the highest possible value. The data c_j and a_j are the values and the weights of the corresponding items and b is the total allowable weight in the available space. For the purpose of obtaining lower and upper bounds we use the well known greedy heuristic which involves ordering the variables in decreasing c_i/a_i and then trying to fill the knapsack in that order. That is, the variables with the largest c_i/a_i ratios are sequentially set equal to 1, until $a_r > b - \sum_{i \in S} a_i$, where S denotes the index set of variables set to 1 by the algorithm. Then x_r and the remaining variables are set equal to 0. Our version of the greedy algorithm stops the insertion at x_r. We note that this is the rounded down linear programming relaxation solution of the problem. Let $\theta = b - \sum_{i \in S} a_i$. It is obvious that if $\theta = 0$, then this solution is optimal. An upper bound on the objective function value is given by $z^u = \sum_{i \in S} c_i + \lambda_r \theta$, where $\lambda_i = c_i/a_i, i = 1, \ldots, n$, because it corresponds to the value of the optimal solution of the linear programming relaxation of KP, called LKP. $z^l = \sum_{i \in S} c_i$ is an obvious lower bound.

Let $\bar{c}_j = c_j - \lambda_r a_j$ for $j \in N = S \cup T \cup \{r\}$ where S and T denote the sets of indices of variables equal to 1 and 0 respectively in the solution obtained by the greedy heuristic described above. The $\bar{c}_j$ are actually the reduced (relative) costs in the simplex tableau corresponding to LKP.

Definition

(i) J is *stable* if

$$\sum_{j \in J} |\bar{c}_j| \geq z^u - z^l \tag{2}$$

and (2) does not hold for any proper subset of J,

(ii) $|J|$ is the *order of stability* of stable subset J;

(iii) Let K be the smallest number such that J or some subset of J is stable for all $J \subseteq N \setminus \{r\}$ and $|J| \leq K$. Then K is the *stability* number of KP .

We note that KP may have no stable subsets or its stability number may be undefined. It is not surprising that a variable belonging to a small stable set is more likely to preserve its greedy solution value in an optimal solution than a variable belonging to a larger stable set or a variable not belonging to a stable set at all. Several elementary results can be established regarding stable subsets and they are listed below:

Proposition 1:

Let $J \subseteq N$ with $|J| \leq K$, K is the stability number of KP , and $X = \{x_1, \ldots, x_n\}$ is an optimal solution to KP. Then

$$\sum_{J \cap S} (1 - x_j) + \sum_{J \cap T} x_j \leq K - 1 \tag{3}$$

Proof:

The upper bound z^u decreases by at least $\bar{c}_j$ if x_j is fixed at 0 for $j \in S$ and at 1 for $j \in T$. If we reverse the values for all variables in a stable set in this manner and fix them, the upper bound will decrease by at least the sum of $\bar{c}_j$'s in that set which is at least as great as $z^u - z^l$ by definition. That is, the upper bound on the objective function value of the resulting problem will fall below the value of the greedy solution. •

Proposition 1 implies that every stable subset contains the index of at least one variable whose greedy solution value and the optimal solution value are the same. In addition,

Corollary 1:

If the stability number of KP is 1, then the greedy solution is optimal.

The following result shows that it is an easy task to find the stability number of the knapsack problem.

Proposition 2:

The stability number K of KP is the solution value to the following knapsack problem:

$$\begin{aligned} K &= \text{Max} \sum_{i \in N \setminus r} y_i + 1 \\ s.t. \quad \sum_{i \in N} |\bar{c}_i| &\leq z^u - z^l \\ y_i &= 0 \text{ or } 1, \ i \in N \end{aligned}$$

and can be found using the greedy heuristic described earlier. If this results in $K = n + 1$, we assume that the stability number is undefined.

The final result establishes a bound on the value of the greedy heuristic.

Proposition 3:

The value of the solution given by the greedy heuristic is at least as large as

$$\lfloor |S|/K \rfloor / (\lfloor |S|/K \rfloor + 1)$$

times the value of the optimal solution of KP .

Proof:

We have $z^l \geq \lfloor |S|/K \rfloor (z^u - z^l)$ because there are at least $|S|/K$ disjoint stable subsets of S, and the sum of $\bar{c}_j$ for each subset is at least as great as $z^u - z^l$, and $z^l = \sum_{j \in S} c_j \geq \sum_{j \in S} \bar{c}_j$. Also $z^u \geq z^l$ and $|S|/K \geq \lfloor |S|/K \rfloor$, proving the above statement. •

If we have $|S| \leq K$, this bound is trivial. For this reason we assume, in the rest of the paper , that we are dealing with problems with $K \leq |S|$. This is not a great loss because stability is a data dependent property, and we are trying to demonstrate its use when it is available.

The bound given above is never tight. We will use it together with the stability results to improve the performance of approximation algorithms. We will also discuss possible ways of using this information for branch and bound algorithms.

3. IMPLICATIONS OF STABLE SUBSETS

3.1 Implication for Polynomial Approximation Algorithms

Sahni [17] gives a "series of increasingly

accurate algorithms" to obtain approximate solutions to the zero-one knapsack problem. He proves that in the worst case, his algorithm can give results no smaller than $L/(L+1)$ times the optimal solution, i.e., $z^l \geq (L/(L+1))z^*$ is true where z^* is the optimal solution value of KP, $L \geq 1$ being the size of the largest combination completely enumerated, and has time complexity $O(Ln^{L+1})$ for $L \geq 1$. Here, z^l is the value of the approximate solution.

Let us consider the following algorithm with the purpose of showing how Sahni's bound can be improved in some cases:

Step 1: Apply the Greedy Algorithm to obtain S and K as defined earlier. Set $1 \leq L \leq K-1$.
Step 2: Set $i = 1$.

Step 3: Set a previously untested combination of i variables in S equal to 0. The remaining variables in S are equal to 1.

Step 4: Fill the rest of the knapsack by, first setting all feasible combinations of variables not in S and with size $1, 2, \ldots \min(L, K-1-i)$ equal to 1, and then filling the remainder by the application of the Greedy Algorithm to the remaining variables.

Step 5: Save the best solution found so far. If all i-combinations are tested in Step 3, go to Step 6, Otherwise return to Step 3.

Step 6: Set $i = i + 1$. If $i \leq L$, return to Step 3. Otherwise stop.

This algorithm has time complexity of $O(L^2|S|^{L+1}|N \setminus S|^{L+1})$. How this bound compares with $O(Ln^{L+1})$, the bound for Sahni's algorithm, depends on the values of $|S|$ and L. These parameters may, of course, be different for problems of the same size.

If we set $L = K - 1$, then the algorithm above finds an optimal solution, because the search process implicitely enumerates all possibilities in that case. That's why it is logical to discuss the degree of approximation (the worst case bound) for $L \leq K - 2$ only. While executing the algorithm we will encounter a situation where $|S| - (K-2) + L$ variables with the greatest c_j values in an optimal solution will be assigned a value of 1. This means a worst case bound of

$$z^l \geq [(|S| - (K-2) + L)/(|S| - (K-2) + L + 1)]z^*$$

in terms of the ε-approximation algorithm described above. But it is also true that

$$[(|S| - (K-2) + L)/(|S| - (K-2) + l + 1)] \geq [(R+L)/(L+1)]$$

for any $L \leq K - 2$ where $R = \lfloor |S|/K \rfloor / (\lfloor |S|/K \rfloor + 1)$ in the above inequality.Thus

$$z^l \geq ((R+L)/(L+1))z^*$$

holds true. It may be argued that R is problem dependent and there is no guarantee its value won't be small. Howewer, the information, whether it is of little or great use, is at our disposal as soon as we set up the problem for the approximation algorithm. To illustrate the point more clearly, let us consider the following instance.

Let $|S| = 25$, $n = |N| = 100$ and the stability number $K = 4$. For $L = 1$, our algorithm guarantees a solution with $z^l \geq .928z^*$. We must note that, regardless of the quality of an available solution, Sahni's algorithm requires a tremendous amount of work to achieve high levels of performance bound. For example, knowing that an initial feasible solution is at least 90 percent of the optimum does not lessen the computational burden of his algorithm if you want, at minimum, 92 percent of the optimum guaranteed. You have to set $L = 12$ to get this required minimum bound. The difference between the computational effort required by our algorithm with $L = 1$ and Sahni's with $L = 12$ is striking. 92% optimum can be achieved in $\approx 7 \times 10^3$ operations with our algorithm, whereas the same bound for the same problem is achieved in 12×10^{26} operations with Sahni's algorithm. In other words we have to spend about $\approx 1/10^{23}$ of the effort required by Sahni's algorithm to achieve the same level of confidence.

3.2 Implication for Fully Polynomial Approximation Algorithms

Ibarra and Kim [9] and Lawler [12] show how dynamic programming can be used to obtain approximate solutions to the knapsack problem in polynomial time. In [13] we show how their scheme can be improved by using a revised version of dynamic programming. In this section, the dynamic programming algorithm given in [13]will be used to introduce the stability number into the complexity functions. A short outline of the dynamic programming mentioned above is appropriate here. Reference [13] can be seen for more detail.

The dynamic programming recursive formulae used for KP are as follows:

$$f_j(b') = \begin{cases} f_{j-1}(b') & \text{if } b' < a_j \\ \max[f_{j-1}(b' - a_j) + c_j; f_{j-1}(b')] & \text{otherwise} \end{cases}$$

for $b' = 0, 1, \ldots, b$, $j = 1, 2, \ldots, n$ and $f_0(b') = 0, b' = 0, 1, \ldots, b$. Whenever $f_j(b') > f_{j-1}(b')$, the index j is stored in a vector $I(b')$. The variable indexed at b^* (the optimal KP usage) is fixed at 1. Then, setting $\hat{b} = b^* - a_{j^*}$, a_{j^*} being the coefficient of the fixed variable, and fixing all variables with larger indices to 0, we divide the remaining variables into two subsets of equal cardinality. Applying the DP recursion to each of these subsets separately, i.e., to each of the two new KP problems with a right hand side $\hat{b}$; we obtain four vectors of dimension $1 \times \hat{b}$, namely: $f'_{n/2}, I'_{n/2}(\hat{b})$, and $f''_{n/2}, I''_{n/2}(\hat{b})$. After determining the value b' in $\{0, 1, \ldots, \hat{b}\}$ where $f'_{n/2}(b') + f'_{n/2}(\hat{b} - b') = f_n(\hat{b})$, we fix two more variables with indices; $I'_{n/2}(b')$, and $I''_{n/2}(\hat{b} - b')$ at 1, and reapply the partitioning and recursion procedure until the solution to the original problem is constructed. This algorithm is of $O(nb \log_2 n)$ time and $O(b)$ space complexity.

Let us assume that we have determined the stability number K for a knapsack problem using the greedy algorithm. This tells us that at most $K - 1$ of the variables in S can be equal to 0 in any optimal solution. This will be used in defining two new knapsack problems:

$$\text{Max } b_1(j) = \sum_{i \in S} a_i x_i, \text{ st: } \sum_{i \in S} c_i x_i = j, \; j = 1, \ldots, (K)C_{max} \tag{4}$$

$$\text{Min } b_2(j) = \sum_{i \in N \setminus S} a_i x_i, \text{ st: } \sum_{i \in N \setminus S} c_i x_i = j, \; j = 1, \ldots, (K)C_{max} \tag{5}$$

where N, S, K are as defined earlier, and $C_{max} = \max c_i$.

The optimal solutions to these problems can be obtained in $O(nKC_{max} \log_2 n)$ time and $O(KC_{max})$ space using the algorithm outlined above. When the rounding down procedure described in [9], [12], and [13] is used for c_j, then these complexities translate into $O(nK^2 \log_2 n/\epsilon)$ time and $O(K^2/\epsilon)$ space, where ϵ is the desired degree of approximation. Note that the positions of the cost coefficients c_j and a_j have been exchanged in the recursion formulae. This is due to the fact that scaling can be carried out only on cost coefficients without affecting feasibility. Smaller c_j values lead to a smaller state space, and thus lesser computational work.

What remains now is the construction of a solution to the original problem from the combination of the solutions to these new knapsack problems. Application of the simple procedure explained below will be sufficient for this derivation.

The solution to problem (4) gives the maximal resource consuming solutions for $j = 1, \ldots, (K)C_{max}$ and that of problem (5) gives the minimal resource consuming solutions for $j = 1, \ldots, (K)C_{max}$. Recalling that the variables of the first problem are in S and second problem in $N \setminus S$ (sets defined by the greedy algorithm), we aim at improving the greedy solution by exchanging some of the variables in S with those in $N \setminus S$. The total number of variables involved in this exchange cannot exceed K by our Proposition 1.

This improvement will be possible if we can find two sets of variables $E^1 \subset S$ and $E^2 \subset (N \setminus S)$ such that

$$\sum_{i \in E^1} c_i < \sum_{i \in E^2} c_i$$

and

$$\sum_{i \in E^1} a_i + \theta \geq \sum_{i \in E^2} a_i$$

where $\theta = b - \sum_{i \in S} a_i$ as before.

The elements of the sets E^1 and E^2, which will bring about maximum improvement, can be obtained from the vectors $b_1(j)$ and $b_2(j)$ by the following procedure.

Order the components of $b_1(j)$ and $b_2(j)$ in increasing $b(j)$ values. Find $b_1(j_1)$ and $b_2(j_2)$ such that $j_2 - j_1 > 0$ is maximum and $b_1(j_1) + \theta \geq b_2(j_2)$. E^1 is the solution corresponding to $b_1(j_1)$ and E^2 is that of $b_2(j_2)$. The optimal solution of the original problem is $S^* = (S \setminus E^1) \cup E^2$. The complexity of the procedure is $O(KC_{max} \log_2(KC_{max}) + KC_{max})$. The first term corresponds to the ordering process and the second is for the search process. When C_{max} is scaled down the

time complexity of this algorithm comes out as

$$O((nK^2 \log_2 n)/\epsilon + K^2/\epsilon + (K^2/\epsilon)\log_2(K^2/\epsilon)).$$

The advantage of this algorithm becomes significant when K, the stability number is small relative to n, the number of variables of the problem. The time complexity of previous approaches would be comparable to the above formula when K is replaced by n.

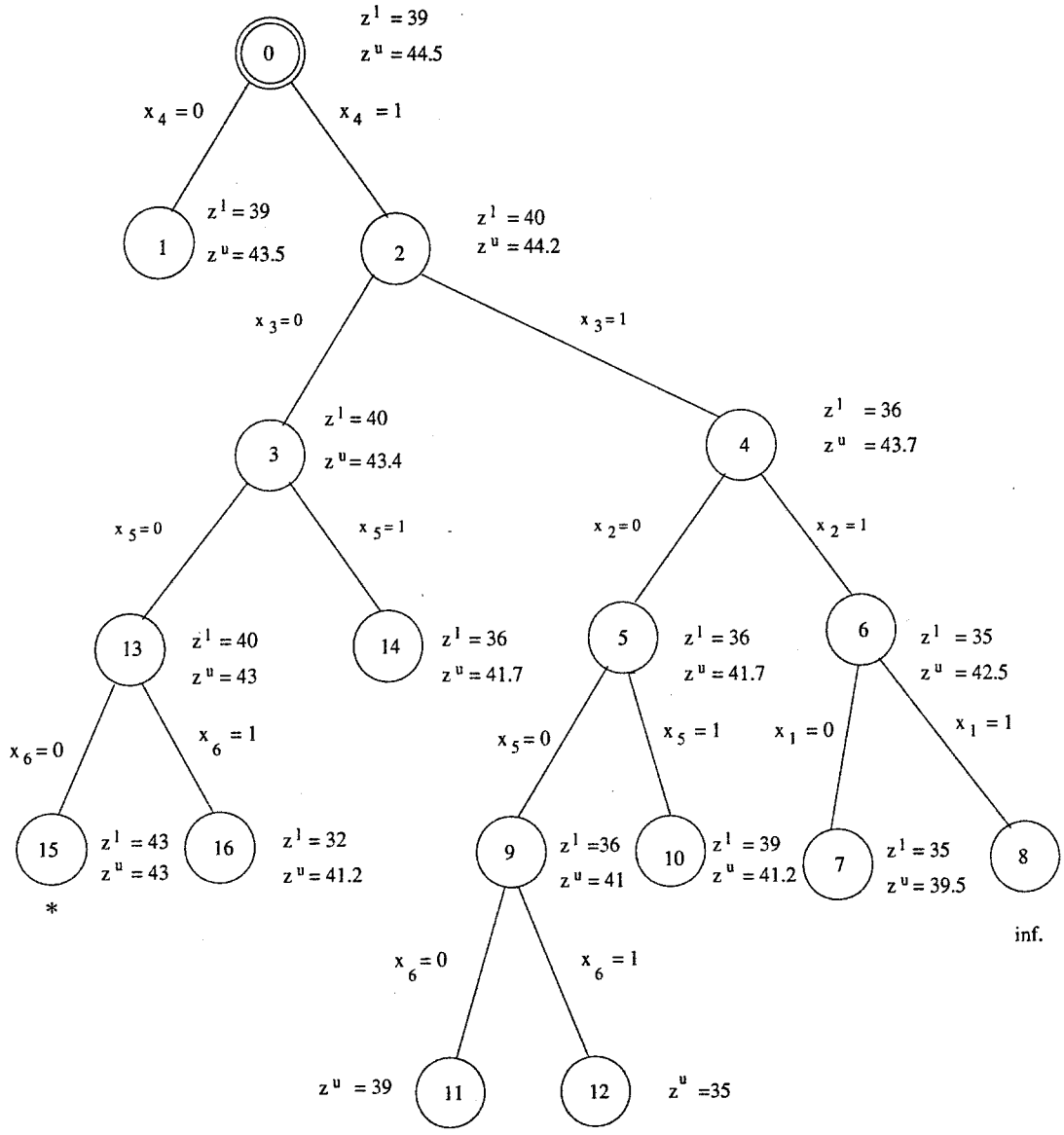


Figure 1: B & B Tree for the Standard Method

3.3 Implication for Branch and Bound Algorithms

We know that branch and bound related enumerative techniques work well on some problems and poorly on others. When they fail, i.e., encounter a difficult problem, the enumeration process may become complete. In the context of zero-one programming this may mean testing

all 2^n possibilities where n is the number of variables of the problem under consideration. Thus, classification of problems into "computationally difficult" and "not so difficult" categories might be quite appropriate. As to how this classification can be done there is no definite answer. Our aim here is to analyze the relationship between the concept of stability and this question.

Any enumerative procedure, like Greenberg and Hegerich's [8] branch and bound algorithm, using $z^u - z^l$ and the reduced costs will immediately fathom nodes with

$$\sum_{j\in S}(1-x_j)+\sum_{j\in N\setminus S}x_j \geq K$$

where $x_j = 0$ or 1 for $j = 1,\ldots,n$. Thus, $O(Kn^K)$ may be considered the time complexity of B & B in terms of a given stabilitly number. If the result of Proposition 1 is used in the process the fathoming test at the tips of the B & B tree can be avoided. This would mean a tree with at most $(K-1)n^{(K-1)}$ branches rather than at most Kn^K branches. Savings in terms of computational effort can be considerable if an expensive fathoming test is being used.

Obviously, a small stability number K indicates a potentially easier problem to solve using enumerative schemes. Use of this fact in branch and bound may be very fruitful as the following example illustrates. Consider the small 0-1 knapsack problem:

$$\text{Max } 15x_1 + 14x_2 + 10x_3 + 11x_4 + 8x_5 + 6x_6 + 3x_7$$

$$\text{s.t. } 8x_1 + 9x_2 + 7x_3 + 8x_4 + 7x_5 + 6x_6 + 3x_7 \leq 28$$

$$x_j = 0 \quad \text{or} \quad 1, \quad i \in \{1,2,\ldots,7\}$$

The straightforward (depth first) application of the B & B algorithm finds the optimal solution after growing a 16 node tree shown in Figure 1. Suppose, instead we change the branching rule as follows:

To pick the initial variable to branch on, solve the LP relaxation of the problem first. Then solve it $(n+1)$ more times, each time fixing a variable to the complement of its value in the initial LP relaxation, and also computing the corresponding stability number K. We have to do this $(n+1)$ times as opposed to n times because the r'th variable can not be complemented like others since it has a fractional value. We have to fix its value twice (once at 0, then at 1) and compute the corresponding K's. The variable giving the smallest K when fixed to either 0 or 1 is used to form the initial branch. For the example given above the lowest K is obtained when x_7 is set equal to 1, so the two branches leading out of the initial node correspond to $x_7 = 0$ and $x_7 = 1$. Further branching is done from the $x_7 = 1$ node using the same criterion, i.e., the LP relaxation of the subproblem resulting from fixing $x_7 = 1$ is solved n times to determine the variable leading to lowest K (stability number). The process continues in this manner until all nodes are fathomed. The tree corresponding to the solution of the example problem given above is shown in Figure 2. The initial solution of the LP relaxation provides the following information:

x_4 is fractional, i.e., $r = 4$.
λ=11/8,
$\theta = c_r x_r = z^u - z^l = 5.5$
$z^u = 44.5$, $z^l = 39.$,
$\bar{c}_1 = 4.$, $\bar{c}_2 = 1.625$, $\bar{c}_3 = .375$, $\bar{c}_5 = -2.625$, $\bar{c}_6 = -2.25$, $\bar{c}_7 = -1.125$, and
$K = 5$: the stability number.

Setting x_1, x_2, x_3, x_4 equal to 0, and x_4, x_5, x_6, x_7 equal to 1, one at a time we see that $x_7 = 1$ leads to a subproblem with $K = 2$. So, the two branches of the initial node correspond to $x_7 = 0$ and $x_7 = 1$. Going for depth first in the branch with the lower K, we see that all variables except x_3 and x_4 may be fixed. The only remaining alternatives are $x_3 = 1$ or $x_4 = 1$. Fathoming both alternatives returns us to the branch $x_7 = 0$ with a good solution, which is verified to be optimal after evaluating 4 more nodes.

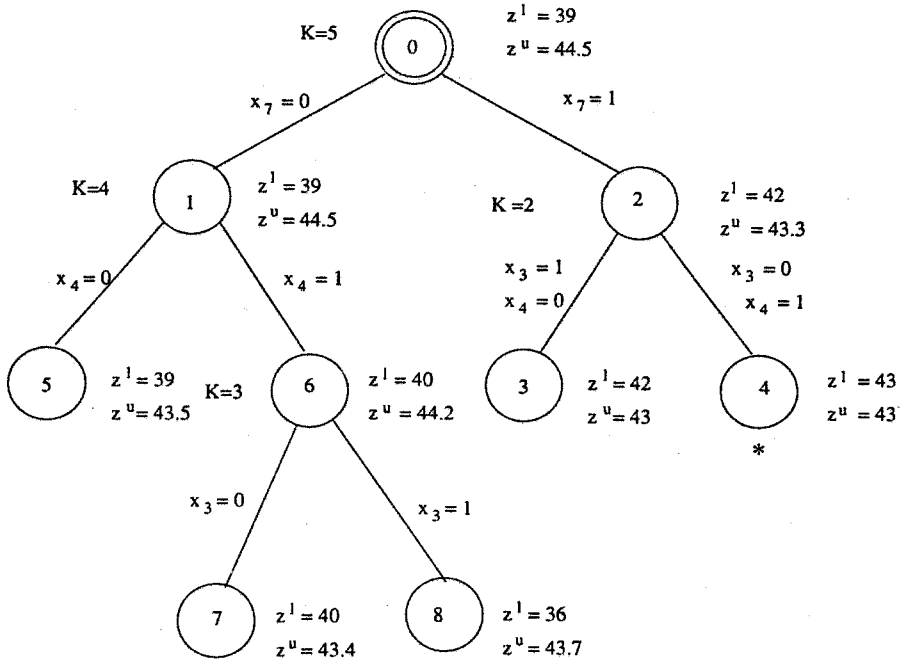


Figure 2: B & B Tree for the Revised Method

Comparison of the two trees Figure 1 and Figure 2, indicates the possible advantage of using the new rule. The only drawback is the amount of extra computations required at each node. There is a trade off between the reduction in the search tree size and the computational load per node. We have run a small experiment in which the increase in the amount of computations per node is kept at minimum by calculating the stability number once per branch only. This is achieved by branching on the fractional variable from each node and using the stability number as the criterion in going for depth instead of the upper bound given by the LP relaxation as it is usually done. In other words, K is computed for $x_r = 0$ and $x_r = 1$, then further branching is carried out from one of these two nodes with the lower K value. The results of comparing this method with the standard B&B (the method used to get the B&B tree in Figure 1) are given in Table 1. The details of the experiment are as follows. Knapsack problems with number

RHS values	$1/3 \sum_i a_i$		$1/2 \sum_i a_i$		$3/4 \sum_i a_i$	
Number of Variables	Standard	Revised	Standard	Revised	Standard	Revised
10	204	188	158	156	86	88
20	425	389	172	166	84	84
30	610	600	222	196	178	180
40	741	693	460	414	206	174

Table 1: Total Number of Nodes Enumerated with the Standard and the Revised B&B Algorithms for some Knapsack Problems.

of variables 10, 20, 30 and 40 are generated using a random number generator from a uniform distribution. The cofficients c_j and a_j range between 1 and 100. The right hand side coefficient b is set equal to 1/3, 1/2 and 3/4 times $\sum_{j=1}^{n} a_j$ for three different trials. Each trial corresponds

to solving 10 randomly generated problems using ten different seeds. A total of 120 problems are solved like this. The standard B&B visits 3546 nodes in this process where as the revised B&B visits 3328 nodes. A saving of approximately 6 percent is obtained at almost no cost.

Besides developing new strategies for branching as described above, the stability number may be used in making "stop or go" decisions in B&B type of algorithms. The stability concept provides instant information regarding how much work may be required to fathom a given branch. That is, it provides a means of making trade offs between computational burden and solution quality. For example, a 1000 variable problem may be considered solvable to optimality if its stability number is small even if the available lower bound on the objective function value is close to the upper bound. On the other hand, the available lower bound may be accepted as satisfactory if the stability number is high, indicating a possibly prohibitive amount of computational work.

In each of the approaches described in this paper the savings are best when K, the stability number, is small compared to n, the number of variables in the problem. Our limited experimental analysis indicates that this is the case. In a set of randomly generated 60 problems ranging in size from 200 to 1000 variables, the stability number ranged from 3 to 63. The stability number seemed to be of the magnitude $O(\log_2 n)$.

4. GENERALIZATION OF STABILITY TO GENERAL ZERO-ONE PROGRAMMING.

The concept of stability in 0-1 integer programming is precisely the same as for the knapsack problem as are the extensions of Proposition 1 and the Corollaries which followed. The difficulty is in finding $\lambda \in R^m$ (where m is the number of constraints) such that we can define S, i.e.,

$$S = \{j \mid c_j - \lambda a_j > 0, \sum_{j \in S} a_j \leq b, \text{ and } c_j - \lambda a_j \leq 0, \forall j \notin S\}$$

where $\lambda, a_j, b \in R^m$. The simplex algorithm with upper bounded variables is one way to obtain λ. Brooks and Geoffrion [2] suggest another approach to find these multipliers in general. The task is much easier for the multidimensional knapsack problem, i.e., when all data are positive. Algorithms presented in [3] and [14] immediately give rise to the needed information.

Most importantly, however, the effort needed to find these values may be worth it as we anticipate similar implications regarding the potential reduction of computational burden when using these concepts in solving these more general problems.

ACKNOWLEDGEMENTS

This research was partially supported by the Natural Sciences and Engineering Research Council of Canada, Grant No. A4124.

BIBLIOGRAPHY

[1] Balas, E. and E. Zemel,"An Algorithm for Large Zero-One Knapsack Problems," *Operations Research*, Vol. 28, no. 5 (1980), pp.1130-1154

[2] Brooks, H. and A. Geoffrion,"Finding Everett's Lagrange Multipliers by Linear Programming." *Operations Research*, Vol. 14, (1976), pp. 1149-1153.

[3] Chandra, A.K.; D.S.Hirschberg, and C.K. Wong, "Approximation Algorithms for Some Generalized Knapsack Problems," *Theo. Comp. Sci.* Vol. 3 (1976), p. 293-304.

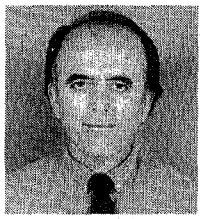
[4] Dembo, R.S. and P.L. Hammer,"A Reduction Algorithm for Knapsack Problems," *Methods of Operations Research*, 36(1980) p. 49-60

[5] Fayard, D. and G. Plateau,"Resolution of the 0-1 Knapsack Problem: Comparison of Methods," *Mathematical Programming*, Vol. 8 (1975), p. 272-307.

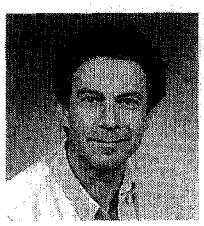
[6] Fayard, D. and G. Plateau,"An Algorithm for the Solution of the 0-1 Knapsack Problem", *Computing*, Vol.28 (1982) p.269-287.

[7] Glover, F., "A Note on Linear Programming and Integer Feasibility," *Operations Research*, Vol. 16, No. 6 (1976), p. 1212-1216.

[8] Greenberg, H. and R.L. Hegerich, "A Branch Search Algorithm for the Knapsack Problem," *Management Science*, Vol. 16 (1970), p. 327-332.
[9] Ibarra,O. and C.Kim,"Fast Approximation Algorithms for the Sum of Subsets Problems," *J. ACM.* Vol. 22 (1975), p. 463-468.
[10] Ingargiola, G.P. and J.F. Korsh,"A Reduction Algorithm for Zero-One Single Knapsack Problems," *Management Science*, Vol. 20, No. 4 (1974), p. 460-463.
[11] Lauriere, M."An Algorithm for the 0-1 Knapsack Problem", *Mathematical Programming*, Vol.14 (1978) p.31-56.
[12] Lawler, E.L., "Fast Approximation Algorithms for Knapsack Problems," *Mathematics of Operations Research*, Vol. 4, No. 4 (1979), pp.339-356.
[13] Magazine, M.J. and O.Oguz,"A Fully Polynomial Time Approximation Algorithm for the 0-1 Knapsack Problem," *European Journal of Operational Research*, Vol. 8 (1981), p. 270-273.
[14] Magazine, M.J. and O.Oguz,"A Heuristic Algorithm for the Multidimensional 0-1 Knapsack Problem," *European Journal of Operations Research*, Vol. 16 (1984), pp.319-326.
[15] Martello, S. and P.Toth, "A New Algorithm for the 0-1 Knapsack Problem," *Management Science*, Vol. 34 (1988) p.633-644.
[16] Nauss, R.M., "An Efficient Algorithm for the 0-1 Knapsack Problem," *Management Science*, Vol. 23, No. 1 (1976), pp.27-31.
[17] Sahni, S., "Approximate Algorithms for the 0-1 Knapsack Problem," *Jnl. of Assoc. Computing Machinery*, Vol. 22 (1975), p. 115-124.



O. Oğuz is an Associate Professor of Industrial Engineering at Bilkent University, Turkey. He obtained B.Sc. and M.Sc. degrees in IE from Middle East Technical University, and Ph.D. in Management Sciences from University of Waterloo, Canada. His research interests are in discrete optimization and scheduling.



Michael Magazine is Professor of Management Sciences at the University of Waterloo. His current interests include scheduling, production and manufacturing. He is a member of CORS, TIMS and ORSA.