Mathematical Programming 31 (1985) 78-105
North-Holland

EFFICIENT ALGORITHMS FOR SOLVING MULTICONSTRAINT ZERO-ONE KNAPSACK PROBLEMS TO OPTIMALITY

Bezalel GAVISH
Graduate School of Management, University of Rochester, Rochester, NY 14627, USA

Hasan PIRKUL
Academic Faculty of Accounting, Ohio State University, Columbus, OH 43210, USA

Received 31 October 1982
Revised manuscript received 24 April 1984

The multiconstraint 0–1 knapsack problem is encountered when one has to decide how to use a knapsack with multiple resource constraints. Even though the single constraint version of this problem has received a lot of attention, the multiconstraint knapsack problem has been seldom addressed.

This paper deals with developing an effective solution procedure for the multiconstraint knapsack problem. Various relaxations of the problem are suggested and theoretical relations between these relaxations are pointed out. Detailed computational experiments are carried out to compare bounds produced by these relaxations. New algorithms for obtaining surrogate bounds are developed and tested. Rules for reducing problem size are suggested and shown to be effective through computational tests. Different separation, branching and bounding rules are compared using an experimental branch and bound code. An efficient branch and bound procedure is developed, tested and compared with two previously developed optimal algorithms. Solution times with the new procedure are found to be considerably lower. This procedure can also be used as a heuristic for large problems by early termination of the search tree. This scheme was tested and found to be very effective.

Key words: Integer Programming, Multiconstraint Knapsack Problem, Surrogate Constraint, Surrogate Duality, Zero–One Knapsack Problem, Knapsack Problem.

Introduction

The multiconstraint 0–1 knapsack problem is encountered when one has to decide how to make efficient use of an entity which consumes multiple resources. This problem appears as a subproblem in large models for allocating processors and databases in a distributed computer system [13, 14]. Other applications of the problem include resource allocation, project selection and cargo loading [38], capital budgeting [41] and stock cutting problems [19]. Despite its wide applicability, relatively few studies have addressed this problem. Most of the research on knapsack problems dealt with the single constraint version. Balas and Zemel [1], Bulfin and Parker [3], Horowitz and Sahni [27], Ingargiola and Korsh [28], Nauss [36], and Salkin and Kluyver [37] are only a few of the researchers who have extensively studied the single constraint knapsack problem.

One of the early references to the multiconstraint knapsack problem is by Gilmore and Gomory [19], which outlines a dynamic programming algorithm. Green [22] proposed extensions to the Gilmore–Gomory algorithm. An algorithm which makes use of dynamic programming as well as heuristics was developed by Weingartner and Ness [41]. Cabot [4] suggested an enumeration method which uses the Fourier–Motzkin elimination method. More recently, Soyster et al. [39] developed an iterative scheme in which linear programs were solved to generate subproblems that were then solved through implicit enumeration. In a recent paper, Shih [38] presented an algorithm which takes advantage of the special structure of the problem in order to develop branch and bound methods for solving the problem.

In this paper, we present an effective solution method for the multiconstraint knapsack problem. The problem is defined in Section 1, different relaxations of the model including Lagrangian, Surrogate and Composite relaxations are introduced, and the theoretical relations between these relaxations are outlined. Section 2 deals with the derivation of multipliers to be used in various relaxations. In Section 2.1, new procedures for finding surrogate multipliers are introduced together with the theory supporting these procedures. Section 2.2 describes how composite multipliers are obtained. Computational experiments with bounds derived from these relaxations are reported in Section 2.3. Section 3 outlines reduction rules and demonstrates the effectiveness of these rules through computational tests. Section 4 outlines a general branch and bound procedure and introduces various separation, branching and bounding rules. Experiments with these rules are described and computational results are presented. Section 5 describes the final algorithm and reports on its computational performance. This algorithm is compared to an algorithm developed by Shih [38]. The effect of early termination of the branch and bound search is illustrated and this strategy is compared to heuristics suggested by Loulou and Michaelides [33].

1. Relaxations of the multiconstraint knapsack problem

The multiconstraint knapsack problem is formulated as:

Problem-IP

$$Z_{\mathrm{IP}} = \mathrm{Max}\{cx \mid Ax \leq b, x \in \{0, 1\}\}$$

where A is an integer nonnegative m by n matrix, b and c are positive integer vectors of size m and n respectively where $m \ll n$. This coupled with the nonnegativeness of A distinguishes this problem from the general integer programming problem.

Various relaxations of Problem-IP are possible. One common relaxation of integer programs is relaxing the integrality constraints. This relaxation will be referred to as Problem-LP and its optimal value will be denoted by Z_{LP}. Lagrangian relaxation has been successfully applied to many combinatorial optimization problems during the last decade. The use of generalized Lagrange multipliers in optimization problems was first suggested by Everett [8]. The ingenious application of Lagrangian relaxation to the traveling salesman problem by Held and Karp [25] spurred an extensive use of this approach. Since then it has been applied to many different problems such as various location problems [7, 18]; the generalized assignment problem [5]; the multiple traveling salesman problem [16] and to a variety of computer communication network design problems [10–12].

A Lagrangian relaxation of Problem-IP is obtained by selecting one of the constraints (e.g., constraint k) as the active constraint, multiplying the remaining constraints by a vector λ, $\lambda \geq 0$, of Lagrange multipliers and adding the product to the objective function, leading to:

Problem-LR

$$L(\lambda) = \text{Max}\{cx + \lambda(\bar{b}_k - B_k x) \mid a_k x \leq b_k, x \in \{0, 1\}\}$$

where B_k is an $m-1$ by n matrix obtained by deleting row k from A, and $\bar{b}_k$ is the corresponding vector obtained by deleting the kth element of b. The bounds generated by Problem-LR depend on the value of λ. The central issue in obtaining tight bounds through this relaxation is to find a set of multipliers λ^* that satisfy

$$L(\lambda^*) = \underset{\lambda}{\text{Min}}\,\{L(\lambda)\}.$$

An alternate relaxation of Problem-IP is obtained by considering the following problem:

Problem-S

$$S(\mu) = \text{Max}\{cx \mid \mu(Ax - b) \leq 0, x \in \{0, 1\}\}.$$

This scheme was first suggested by Glover [21] and is known as surrogate relaxation. Due to initial disappointing computational experience with this approach, it did not receive much attention in the past. In recent years there has been some renewed interest in surrogate relaxation by Karwan [29] and more recently by Dyer [6]. As with Lagrangian relaxation, the critical issue with this approach is to find a vector of multipliers μ^* that will satisfy the relation:

$$S(\mu^*) = \underset{\mu}{\text{Min}}\{S(\mu)\}.$$

Finally, by combining Lagrangian and Surrogate relaxations, the following composite relaxation is derived:

Problem-C

$$C(\mu, \lambda) = \text{Max}\{cx - \lambda(Ax - b) \mid \mu(Ax - b) \leq 0, x \in \{0, 1\}\}.$$

This relaxation was first introduced by Greenberg and Pierskalla [24]. Again, the objective is to find two sets of multiplers $\tilde{\lambda}$ and $\tilde{\mu}$ that satisfy:

$$C(\tilde{\lambda}, \tilde{\mu}) = \underset{\lambda,\mu}{\text{Min}}\{C(\lambda, \mu)\}.$$

The relationships between these relaxations are well known for general integer programming problems and are summarized in the following lemma.

Lemma 1. *The following inequalities hold between the different relaxations*:

$$Z_{\text{LP}} \geqslant L(\lambda^*) \geqslant S(\mu^*) \geqslant C(\tilde{\lambda}, \tilde{\mu}) \geqslant Z_{\text{IP}}.$$

The relationship between the linear programming bound and the Lagrangian relaxation bound was shown by Geoffrion [17]. Greenberg and Pierskalla [24] and Glover [21] have shown that $L(\lambda^*) \geqslant S(\mu^*)$. Finally, the relationship between the composite and surrogate relaxations was given by Karwan and Rardin [31].

2. Derivation of multipliers

The critical issue in the effective use of Lagrangian, Surrogate and Composite relaxations is the derivation of a good set of multipliers. For a given set of multipliers each of these relaxations reduces to a single constraint knapsack problem which can be solved using any one of several efficient algorithms. A few of these algorithms are the ones developed by Balas and Zemel [1], Horowitz and Sahni [27], Martello and Toth [35] and Nauss [36].

Subgradient optimization methods [26, 10–12] and various multiplier adjustment methods [7, 9] have been found as effective heuristics for obtaining good multipliers. The computational experience with these methods shows that subgradient optimization procedures tend to take longer time but are relatively stable and generate better bounds. This is an iterative method which generates a sequence of multipliers using the following rule:

$$\lambda^{k+1} = \lambda^k + t_k(AX^k - b)$$

where X^k is an optimal solution to $L(\lambda^k)$ and t_k is a positive scalar step size. The Lagrangian bounds reported in this paper are computed using a subgradient optimization method with the following step size:

$$t_k = \delta_k \cdot (L(\lambda^k) - Z_f)/\|Ax^k - b\|^2$$

where Z_f is a feasible solution value and δ_k is a scalar satisfying $0 \leqslant \delta_k \leqslant 2$. The sequence δ_k is determined by setting δ_0 to 2 and halving δ_k whenever the bound fails to improve in the last 15 iterations. The active constraint is chosen to be the one that produces the lowest bound when the problem is treated as a single constraint knapsack problem by ignoring the rest of the constraints.

2.1. New procedures for computing surrogate multipliers

Considerably less experience has been accumulated with the Surrogate relaxation. The studies by Glover [20, 21], Greenberg [23, 24], Leunberger [34], Karwan and

Rardin [30, 31], and Dyer [6] constitute the bulk of the research done in this area. Search procedures to find Surrogate multipliers were proposed by Karwan and Rardin [30] and Dyer [6]. We propose alternate search procedures and report on the empirical computational experience with these procedures.

In [21], Glover outlined an NP-complete procedure which could be used to find optimal multipliers for a two constraint problem. In this section, a new algorithm is proposed to the two constraint knapsack problem. It is based on the following surrogate problem.

$$S(\psi, \mu) = \operatorname{Max}\{cx \mid (\psi, \mu) \cdot (Ax - b) \leqslant 0,\ x \in \{0, 1\}\}$$

where ψ and μ are scalars and A is a 2 by n matrix.

It is assumed that the two constraints are linearly independent and that constraint one is violated by the solution of $S(0, 1)$ and constraint two is violated by the solution of $S(1, 0)$ (otherwise an optimal solution to the Surrogate problem is found). The following lemmas are proved in [15].

Lemma 2. *Let x^* be an optimal solution to $S(1, \mu)$. x^* can violate, at most, one constraint of the two-constraint problem.*

Lemma 3. *Let μ' be the smallest value of μ such that the solution of $S(1, \mu)$ satisfies the second constraint. Then for μ_1 and μ_2 such that $0 < \mu_1 < \mu_2 < \mu'$, $S(1, \mu_1) \geqslant S(1, \mu_2)$.*

Lemma 4. *Let μ'' be the smallest value of μ such that the solution of $S(1, \mu)$ does not satisfy the first constraint. Then for μ_1 and μ_2, such that $\mu'' < \mu_1 < \mu_2 < \infty$, $S(1, \mu_1) \leqslant S(1, \mu_2)$.*

From those Lemmas, it follows that when $\mu' \neq \mu''$, any value of μ in the range $(\mu'; \mu'')$ is an optimal multiplier. If $\mu' = \mu''$, then the optimal multiplier is equal to $\mu' = \mu''$. These observations are used in the following algorithm for computing an ε-neighborhood of an optimal multiplier.

Procedure-1 (Computing an ε-neighborhood of optimal multipliers)

Step 1. Let μ_H and μ_L be two multiplier values such that the solution of $S(1, \mu)$ satisfies the first constraint at $\mu = \mu_L$ and does not satisfy it at $\mu = \mu_H$.

Step 2. If $\mu_H - \mu_L \leqslant \varepsilon$ STOP. (An ε-range which encloses an optimal multiplier has been determined.)

Step 3. Let $\mu = \mu_L + (\mu_H - \mu_L)/2$. Solve $S(1, \mu)$. If in the solution to $S(1, \mu)$

(i) both constraints are satisfied STOP. (An optimal solution has been obtained.)

(ii) only the first constraint is satisfied, let $\mu_L = \mu$, go to Step 2.

(iii) the first constraint is not satisfied, let $\mu_H = \mu$, go to Step 2.

The algorithm starts with an initial set of multipliers and uses a Bisection procedure to reduce the gap between the multipliers.

The following numerical example is used to demonstrate the shape of the step function $S(1, \mu)$ (see Figure 1) and the steps taken by the algorithm.

$$c = \{45 \quad 57 \quad 43 \quad 44 \quad 61 \quad 35 \quad 41 \quad 54 \quad 18 \quad 22 \quad 78\},$$

$$A = \begin{Bmatrix} 94 & 24 & 69 & 95 & 63 & 2 & 98 & 29 & 15 & 9 & 100 \\ 26 & 91 & 35 & 15 & 80 & 71 & 20 & 76 & 33 & 46 & 86 \end{Bmatrix},$$

$$b = \{351, 192\}.$$

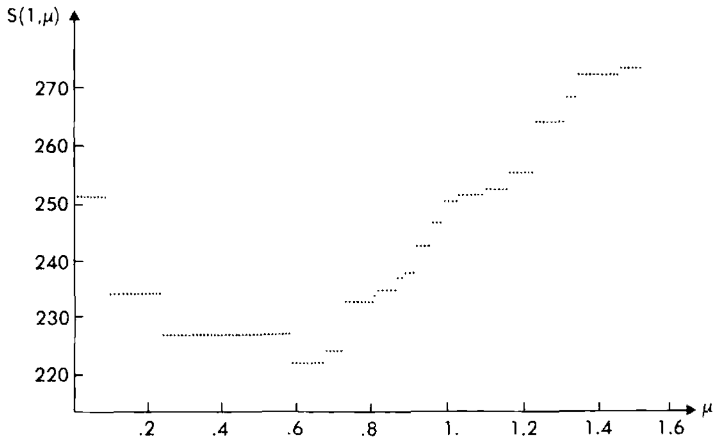

Fig. 1. The function $S(1, \mu)$ plotted as a function of μ.

The second constraint is the tighter of the two constraints (produces a lower objective function value when it is combined with vector c to form a single constraint knapsack problem); therefore, the variable multiplier μ is assigned to the first constraint. Table 1 illustrates the application of the algorithm to this problem.

Table 1
Bisection cuts in Procedure-1

Iter.	μ_L	μ_H	Interval size	Objective function	Constraint violated	Bisection cut
1	0.5000	1.000	0.5000	233	2	0.7500
2	0.5000	0.750	0.2500	222	2	0.6250
3	0.5000	0.625	0.1250	227	1	0.5625
4	0.5625	0.625	0.0625	222	2	0.5937

Procedure-1 can be incorporated into a heuristic for computing multipliers to problems having more than two constraints. This method identifies a descent direction. A search along this direction is done using Procedure-1 while the other multipliers are kept constant. The resulting set of multipliers is used to identify a new direction and the search is repeated along this direction. The following Lemma identifies a potential search direction.

Lemma 5. *Let x' be a solution of $S(\mu)$ and x'' be a solution of $S(\mu + \varepsilon e_j)$ where $\varepsilon > 0$ and e_j is a unit vector. If $a_jx' > b_j$ and $a_jx'' > b_j$ then an increase in the j'th element of μ could possibly improve the objective function value.*

Proof. Define a two-constraint problem where the first constraint is $(\mu A)x \leq \mu b$ and the second constraint is $a_jx \leq b_j$. The proof follows from Lemma 3.

From Lemma 5 follows that all the constraints violated by the current solution point to potential search directions. Among these candidates the most violated constraint is selected for generating the search direction. The method is outlined in Procedure-2.

Procedure-2 (Computing surrogate multipliers for a multiconstraint problem)

Step 1. Determine the constraint which produces the lowest objective function value when all other constraints are ignored in Problem-IP. Renumber this constraint as constraint 1.

Step 2. Determine the constraint most violated by the solution of the single constraint knapsack problem which is formed by the original objective function vector c and constraint 1. Call this constraint 2.

Step 3. Apply Procedure-1 to the problem determined by constraints 1 and 2. If the objective function does not decrease in Procedure-1, or if the number of iterations reached an upper limit, STOP.

Step 4. Let the surrogate constraint determined in Step 3 be constraint 1. Go to Step 2.

At every iteration of Procedure-1, a single constraint 0–1 knapsack problem is solved. This is the time-consuming step of the procedure. It is possible to improve this step by replacing these 0–1 knapsacks with continuous knapsacks. This modified version of Procedure-1 is denoted as Procedure-1A and is no longer guaranteed to find an ε-neighborhood of an optimal surrogate multiplier for Problem-IP. In fact, the continuous knapsack formed using the new multiplier solves the Linear Programming relaxation (Problem-LP) of Problem-IP. The modified version of Procedure-2 which uses Procedure-1A in Step 3 is denoted by Procedure-2A and can be viewed as a heuristic that finds surrogate multipliers for Problem-IP. But, it can also be viewed as a heuristic that attempts to solve Problem-LP.

Lemma 6. *The solution to the continuous relaxation of the surrogate problem which is formed by using the optimal dual variables of Problem-LP as multipliers is also an optimal solution for Problem-LP.*

Lemma 6 implies that Procedure-2A is a heuristic that searches for optimal dual variables of Problem-LP. It is a simple exercise to show that the surrogate bound obtained using the dual variables of Problem-LP as multipliers is always equal to or tighter than the linear programming bound. Therefore, it is expected that in many instances, the multipliers found by Procedure-2A will also produce bounds that are tighter than the one generated by linear programming. Furthermore, it can be shown [15] that for a given ε value Procedure-1A has a time complexity $O(n \log(\varepsilon))$ and Procedure-2A $O(n \cdot \text{Max}\{\log(\varepsilon); m\})$.

2.2. Derivation of composite multipliers

The directional search methods developed for finding Lagrangian and surrogate multipliers were based on the fact that $L(\lambda)$ and $S(\mu)$ are convex and quasi-convex functions of λ and μ, respectively. In [31], it has been shown that composite relaxation does not have either of these two characteristics and similar search methods are bound to fail in finding optimal or near optimal multipliers.

A heuristic was developed for determining multipliers for composite relaxation. It starts with an initial set of Lagrange multipliers which are equal to zero and searches for the best set of surrogate multipliers. Then, surrogate multipliers are used to form a surrogate constraint and a set of Lagrange multipliers are determined. Procedure-2 or Procedure-2A can be used for computing the surrogate multipliers. A subgradient optimization procedure or a dual descent procedure can be applied for determining the Lagrange multipliers. The dual descent consists of fixing all but one element of the Lagrange multipliers vector to their last value and searching for the best value of the remaining one. Since this is a one-dimensional search over a convex function, a straight-forward search will locate the desired multiplier. This procedure is supported by the observation that for the problems under consideration (with few constraints and $m \ll n$), frequently the solution to the surrogate problem violates only one constraint, leading effectively to a single descent direction. If more than one constraint is violated, then the most violated one is selected for determining the search direction.

2.3. Computational experiments and comparison of bounds

In order to evaluate the quality of the bounds generated by the different relaxations, a set of computational experiments were conducted. Twenty-four groups of problems consisting of 100 or 200 variables and three or five constraints were generated. Each group contained ten problems of the same type. The entries of the coefficient matrix A and the objective function vector c were drawn $U(1, 1000)$. The right-hand side vector elements (b) were determined by summing each row of A and dividing this sum by a constant K equal to 2 or 4.

The quality of a bound is measured by the relative gap between the optimal value and the lower bound value. The effectiveness of any bounding procedure is also related to the effort needed to compute it. In Table 2, an attempt is made to capture both of those factors. The tables contain two rows for each set of problems. The first row reports the minimum, mean and maximum of the relative gap between the optimal solution value and the bound generated. Whenever appropriate, the second row reports the minimum, mean and maximum number of knapsack problems solved in computing that particular bound. The second row entries may be used as an indication of the relative computational effort needed to compute different bounds. Seven bounding methods were tested.

Method 1. The linear programming relaxation of Problem-IP is solved using a simplex algorithm.

Method 2. A surrogate constraint is determined using Procedure-2*A*. The single constraint continuous knapsack problem formed by the surrogate constraint is solved to generate the bound. This bound will at best be equal to the linear programming bound.

Method 3. Problem-LR is solved using a subgradient optimization procedure.

Method 4. This method is similar to Method-2. Again, a surrogate constraint is determined using Procedure-2*A*. But this time a 0–1 knapsack problem is solved instead of the continuous one.

Method 5. A surrogate bound is determined using Procedure-2.

Method 6. In this method Problem-*C* is solved using the heuristic introduced in Section 2.2. Surrogate multipliers are determined using Procedure-2. The dual descent method outlined previously is used to determine the Lagrange multipliers.

Method 7. This method is similar to Method-6 with one difference. A subgradient optimization procedure is used instead of the dual descent procedure.

The bound generated by Method-2 requires the least amount of computing time. The quality of bounds generated by this method is generally comparable to those generated by Method-1. In fact, in many instances these bounds are identical, pointing to the fact that Procedure-2*A* effectively solves Problem-LP. Method-2 is favored over Method-1 since these problems tend to be highly degenerate, leading to an excessive number of simplex pivots. In fact, the simplex code developed by Land and Powell [32] did not solve some problems after 2000 pivots.

Bounds generated by Method-4 are considerably tighter than those generated by Method-2. Fifteen to sixty percent of the gap resulting from Method-2 is closed by Method-4. This method becomes very attractive when we consider the fact that the additional work over Method-2 is the solution of a single constraint knapsack problem. Furthermore, bounds generated by this method were comparable to surrogate bounds produced by Method-5, which requires significantly more computing times as pointed out by the number of knapsack problems solved.

Method-3 which generates Lagrangian bounds did better than Method-1 and Method-2, but performed poorly when compared to the other methods. This is due to the relative slow convergence of the subgradient method.

Composite bounds generated by Method-6 and Method-7 are slightly better than surrogate bounds generated by Method-4. Ten to twenty percent of the gap resulting from Method-4 was closed by composite bounds. This gain has to be evaluated against the extra computational effort needed to attain it (20 to 100 additional knapsack problems). Generally, Method-7 generates better bounds than Method-6.

Table 2
Comparing the quality of bounds and computational effort required by the different bounding methods

Problem data	Method-1			Method-2			Method-3			Method-4		
n, *m*, *K*	Min.	Mean	Max.	Min.	Mean	Max.	Min.	Mean	Max.	Min.	Mean	Max.
100, 3, 2	0.233	0.317	0.414	0.237	0.330	0.425	0.214	0.329	0.460	0.167	0.267	0.374
				28	42.5	77	55	60.4	68			
100, 3, 4	0.428	0.696	1.072	0.429	0.713	1.072	0.431	0.665	0.928	0.273	0.581	0.903
				28	41.5	77	45	68.1	103			
100, 5, 2	0.275	0.493	0.740	0.294	0.566	0.813	0.493	0.732	1.170	0.236	0.505	0.730
				47	70.1	113	34	48.5	73			
100, 5, 4	0.639	1.027	1.312	0.749	1.177	2.172	0.742	1.227	1.740	0.487	0.955	1.517
				48	91.4	184	43	54.7	62			
200, 3, 2	0.093	0.125	0.150	0.094	0.129	0.150	0.089	0.138	0.211	0.056	0.100	0.125
				29	59.7	100	48	57.8	75			
200, 3, 4	0.174	0.247	0.328	0.174	0.255	0.338	0.162	0.267	0.431	0.152	0.202	0.250
				28	44.2	68	48	56.7	66			
200, 5, 2	0.134	0.192	0.263	0.149	0.205	0.288	0.167	0.282	0.369	0.126	0.180	0.249
				56	77.3	119						
200, 5, 4	0.351	0.425	0.539	0.396	0.468	0.582	0.445	0.559	0.861	0.308	0.418	0.527
				49	100.3	191	36	48.4	64			

Problem data	Method-5			Method-6			Method-7		
n, m, K	Min.	Mean	Max.	Min.	Mean	Max.	Min.	Mean	Max.
100, 3, 2	0.134	0.257	0.350	0.088	0.241	0.340	0.071	0.230	0.342
	20	34.2	46	31	44.8	57	30	111.9	196
100, 3, 4	0.061	0.469	0.816	0.057	0.437	0.777	0.061	0.454	0.816
	28	39.1	52	38	50	63	63	81.1	142
100, 5, 2	0.228	0.473	0.675	0.209	0.459	0.667	0.228	0.440	0.675
	37	50.4	70	47	61.2	80	59	108	190
100, 5, 4	0.440	0.902	1.378	0.437	0.878	1.358	0.403	0.873	1.378
	46	57	72	57	67.5	82	69	111.7	154
200, 3, 2	0.056	0.094	0.146	0.051	0.090	0.139	0.056	0.091	0.113
	28	36.5	43	39	47.1	56	46	83	156
200, 3, 4	0.122	0.201	0.252	0.113	0.192	0.250	0.122	0.197	0.252
	28	36.1	52	37	47.5	63	63	77	139
200, 5, 2	0.120	0.173	0.226	0.119	0.168	0.218	0.120	0.171	0.226
	46	59.3	73	56	69.7	83	59	102.5	214
200, 5, 4	0.306	0.429	0.546	0.300	0.422	0.538	0.308	0.415	0.546
	48	61.1	113	59	72.8	124	83	119.1	200

But the subgradient procedure used in Method-7 requires many more knapsack problems to be solved compared to the dual descent procedure of Method-6.

The quality of the bounds generated seems to be insensitive to changes in the range of the uniform distribution from which the coefficients of A and c were drawn. Changes in the right-hand side vector b does have a significant effect. As b_i decreases relative to $\sum_{j=1}^{n} a_{ij}$ the gap between the bound and the optimal solution value tends to increase. This can be observed by comparing the results obtained for K values of 2 and 4.

3. Sensitivity analysis and problem reduction

Many of the efficient algorithms for solving single constraint knapsack problems owe their success to the fact that the majority of variables can be a priori fixed using a reduction procedure leaving a very small problem for the branch and bound phase [28, 35, 36, 1]. The reduction efficiency of the sensitivity analysis is directly related to the tightness of the lower bounds used in the analysis. Since bounds for the multiple-constraint knapsack problem are not as tight as bounds for the single-constraint case, the reduction is not expected to be as effective.

Letting x^* be the solution of $S(\mu)$, we can define $S(\mu \mid x_i = 1 - x_i^*)$ to be the problem obtained by fixing variable x_i to $1 - x_i^*$ where x_i^* is the i'th element of x^*. If $\bar{S}(\mu)$ and $\bar{S}(\mu \mid x_i = 1 - x_i^*)$ are two new problems formed by dropping the integrality requirements of $S(\mu)$ and $S(\mu \mid x_i = 1 - x_i^*)$, then the following penalties can be defined:

$$\delta_{i_1} = \bar{S}(\mu \mid x_i = 1 - x_i^*) + x_i^* c_i - Z_F, \qquad \delta_{i_2} = S(\mu \mid x_i = 1 - x_i^*) + x_i^* c_i - Z_F$$

where Z_F stands for the value of the best known feasible solution.

We calculate δ_{i_1} first. If δ_{i_1} is less than or equal to zero then the variable x_i is fixed to x_i^*. Only after this test has been completed δ_{i_2} is computed and further reduces the problem size. Tables 3 and 4 report on the results of experiments with these reductions. Each entry in these tables represents an average over ten problems that were generated by the same method used for generating the problems in Table 2.

Tables 3 and 4 represent two extreme cases which may be regarded as the best and worst cases for evaluating the reduction effectiveness. Table 3 reports on reductions in which the optimal value of each problem was used as the cutoff point. These are the best possible results that can be expected from these reduction procedures. The cutoff points for Table 4 were set at a fixed percent below the upper bounds obtained from Problem-S so as to include the optimal values of all problems in that particular group. These percentages were determined by solving a sample of problems to optimality and picking the maximum deviation from optimality for each group of problems as a cutoff point. This is a conservative estimate of the cutoff point because in many cases, feasible solutions which lie within these cutoff

Table 3

Reduction with optimal values as cutoff points. (1) Percent of variables fixed at the end of first test. (2) Percent of variables fixed at the end of second test

Right hand side vector (b_i)	Number of variables (n)	Number of constraints (m)	A and c are drawn from					
			$U(1, 10)$		$U(1, 100)$		$U(1, 1000)$	
			(1)	(2)	(1)	(2)	(1)	(2)
$\sum_j a_{ij}/2$	100	3	95.5	97.4	73.9	78.9	69.8	76.0
		5	85.1	86.3	57.7	63.4	52.1	58.4
	200	3	98.8	99.6	85.1	88.7	79.2	83.9
		5	88.9	89.4	69.2	72.8	66.8	71.1
$\sum_j a_{ij}/4$	100	3	89.2	93.1	70.0	77.2	68.0	76.6
		5	64.6	69.0	48.7	52.7	51.2	57.4
	200	3	96.6	97.4	78.4	82.4	76.4	80.4
		5	81.6	83.1	59.4	62.9	59.2	63.2

Table 4

Reduction with cutoff point set at a fixed percent below the initial upper bound. (1) The percent of the upper bound which is subtracted from the upper bound to obtain the cut off point. (2) Percent of variables fixed at the end of first test. (3) Percent of variables fixed at the end of second test

Right hand side vector (b_i)	Number of variables (n)	Number of constraints (m)	A and c are drawn from								
			$U(1, 10)$			$U(1, 100)$			$U(1, 1000)$		
			(1)	(2)	(3)	(1)	(2)	(3)	(1)	(2)	(3)
$\sum_j a_{ij}/2$	100	3	0.27	65.7	66.5	0.34	65.2	71.4	0.38	62.5	69.7
		5	0.49	56.1	56.6	0.60	46.3	50.9	0.73	37.6	43.3
	200	3	0.13	66.1	74.1	0.11	79.6	84.2	0.13	74.7	79.3
		5	0.14	63.0	65.8	0.25	55.7	60.4	0.25	56.8	60.9
$\sum_j a_{ij}/4$	100	3	0.44	70.1	70.9	0.86	56.0	62.6	0.91	55.5	62.3
		5	1.26	38.1	41.5	1.64	29.1	33.7	1.52	31.8	37.9
	200	3	0.21	71.8	76.1	0.28	72.0	75.2	0.25	73.0	76.9
		5	0.42	52.6	55.7	0.50	53.4	57.3	0.53	51.6	55.1

points can be determined without difficulty. Actually, branch and bound procedures tend to locate an optimal or a near optimal solution early in the search tree, making it possible to reduce the problem further, using penalities calculated during the sensitivity analysis. The strategy adopted in the final implementation was to use a conservative estimate during the initial reduction procedure and to change it when a better solution was generated by the branch and bound procedure.

The tables clearly indicate that a significant portion of the original problem can be reduced through sensitivity analysis. Most of the reduction is the result of the first test which requires negligible computing resources. The correlation between the effectiveness of the reduction and the tightness of the bound is clearly reflected in the tables. For a fixed number of constraints, when the number of variables increases, the portion of variables that are eliminated increases, reflecting the tightening of the bound observed in Table 2. In other words, the rate of increase in the reduced problem size is lower than the increase in the original problem size. A similar phenomenon was observed by Balas and Zemel [1] for the single knapsack problem (KP). They defined a core problem as, the subproblem in those variables, whose cost/weight ratio falls between the maximum and the minimum c/a ratio for which x has a different value in an optimal solution to KP from that in an optimal solution to LKP, where LKP is the linear programming relaxation of KP. For the KP the size of the core problem was a small fraction of the full problem size and almost independent of the problem size. A core problem can be defined for the multiconstraint knapsack problem by treating the surrogate constraint as the single constraint. A set of core problems were identified in this fashion to demonstrate the relation of the core problem size to the size of the original problem. Eighteen problem sets consisting of ten problems each were used in this experiment. The results are summarized in Table 5. The objective function vector c and the coefficient matrix A were drawn from $U(1, 100)$. For every problem set, the number of variables, constraints and the formula used in generating the right-hand side vector b are indicated. The results clearly demonstrate that for the multiconstraint knapsack problem the size of the core increases at a very slow rate with problem size, and that beyond a certain size, it seems to be almost independent of the problem size.

No clear relation has been observed between the interval of the uniform distribution from which A and c are drawn and the effectiveness of the reduction procedures. As the right-hand side decreases relative to $\sum_j a_{ij}$ the effectiveness of reduction procedures is reduced. This is to be expected since decreases in the right-hand side lead to a wider gap between the bound and the value of the optimal solution.

4. Branching and bounding strategies

Despite the effectiveness of sensitivity analysis in reducing the size of the problem at hand, reduced problems are still of considerable size. Consequently, an efficient branch and bound algorithm is required to solve these reduced problems. A general branch and bound algorithm was developed and coded to experiment with different branching, separation and bounding schemes. First, this general algorithm is presented, and then, each step is explained in detail. It is assumed that before entering the branch and bound stage, sensitivity analysis is carried out and a new, reduced Problem-IP is formed. It is also assumed that the penalities computed during the sensitivity analysis step are available for use in the algorithm. A block diagram of the algorithm is given in Figure 2.

Procedure-B & B

I. *Initialization*—Let x^* be the best known feasible solution and Z^* be the corresponding objective function value. Define S as the set of all free variables, S_F

Table 5

Size of the core problem versus the size of the original problem, ($A \sim U(1, 100)$, $c \sim U(1, 100)$, $m = 3$, $R \sim U(0, 1)$)

Number of variables in the original problem	100			200			300			400			500			600		
Number of variables in the core problem	Min.	Mean	Max.	Min.	Mean	Max.	Min.	Mean	Max.	Min.	Mean	Max.	Min.	Mean	Max.	Min.	Mean	Max.
$b_i = \sum_j a_{ij/2}$	1	7.6	11	4	11.2	19	5	15.4	26	8	15	28	10	17.1	27	16	18.6	23
$b_i = \sum_j a_{ij/4} + R^* \sum_j a_{ij/4}$	5	9	14	6	14	22	6	14.6	25	4	14.8	32	1	16.3	33	6	15.2	30
$b_i = \sum_j a_{ij/4}$	3	9.1	15	5	12.4	21	10	15.7	30	8	17.3	27	16	18.7	34	17	21.3	31

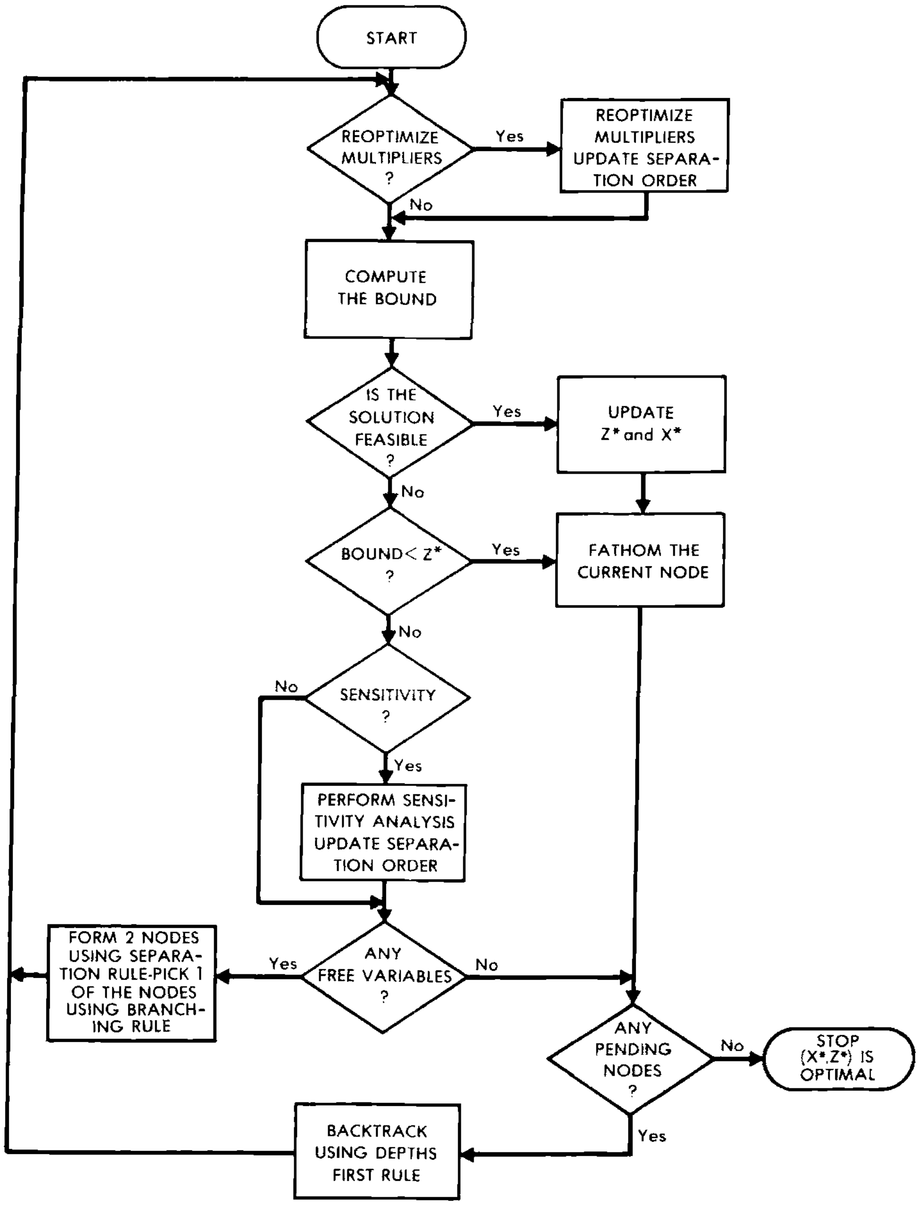
START
REOPTIMIZE MULTIPLIERS ?
Yes
No
REOPTIMIZE MULTIPLIERS UPDATE SEPARATION ORDER
COMPUTE THE BOUND
IS THE SOLUTION FEASIBLE ?
Yes
No
UPDATE Z* and X*
BOUND< Z* ?
Yes
No
FATHOM THE CURRENT NODE
SENSITIVITY ?
No
Yes
PERFORM SENSITIVITY ANALYSIS UPDATE SEPARATION ORDER
ANY FREE VARIABLES ?
Yes
No
FORM 2 NODES USING SEPARATION RULE-PICK 1 OF THE NODES USING BRANCHING RULE
ANY PENDING NODES ?
No
Yes
STOP (X*,Z*) IS OPTIMAL
BACKTRACK USING DEPTHS FIRST RULE

Fig. 2. Block diagram of Procedure-*B & B*.

as the set of all fixed variables, and P as the set of all pending (not fathomed) nodes. Let $s=\{\text{all variables}\}$, $S_F=\emptyset$, and $P=\{\text{node } 0\}$.

II. *Separation*—If $P=\emptyset$ GO TO (VI). If $S=\emptyset$ GO TO (III). Otherwise, choose a variable x_j according to a separation rule. Generate two new nodes corresponding to $x_j=0$ and $x_j=1$. Update S, S_F and P accordingly. If fixing x_j to 1 leads to infeasibility, exclude the corresponding node from P.

III. *Branching*—Choose a node from the set P according to the branching rule.

IV. *Bounding*—Decide if a new bound should be computed and if so, the type of bound computed according to the bounding scheme being used. If no bound is computed, GO TO (II). If a new bound is computed and the corresponding solution is feasible for Problem-IP, fathom the node and update P. If the bound is less than Z^*, update Z^* and x^* accordingly, GO TO (III). If the solution is not feasible and the bound is above Z^*, fathom the node, update P and GO TO (III). If the bound is less than Z^* continue.

V. *Sensitivity Analysis*—Decide if sensitivity analysis is required. If so, carry out the analysis and update S, S_F and P. Update the separation order, GO TO (II).

VI. *Termination*—x^* is the optimal solution and Z^* is the corresponding objective function value.

Bounding Rules—At any node of the branch and bound tree four bounding options are considered:

(1) Not to compute a new bound;

(2) To solve $\bar{S}(\mu)$ with the most current multiplier set;

(3) To solve $S(\mu)$ with the most current multiplier set;

(4) To calculate a new set of surrogate multipliers μ^* using Procedure-2A and to solve $S(\mu^*)$.

Combinations of these options are used as the basis for the experimental algorithms reported in this paper. Since for a given multiplier vector μ', $\text{Min}_\mu\ S(\mu) \leq S(\mu') \leq \bar{S}(\mu')$, it is clear that, as the recomputing frequency of the multiplier vector μ increases, the number of nodes in the branch and bound tree generally decreases. Thus, a tradeoff has to be made between the time spent for computing multipliers and the time spent for handling a larger branching tree.

It is not necessary to calculate a bound at every node of the tree since the nodes which will not result in improved bounds can be detected a priori. If x^* is the solution of $S(\mu^*)$, when a new node is created by fixing variable x_i to x_i^*, then the bound computed in this new node using μ^* will be the same as the bound of the predecessor node since the solution of the restricted problem $S(\mu^* | x_i = x_i^*)$ will be equal to $x^* = \{x_i^*\}$. A similar argument holds for the bound computed by solving the continuous problem $\bar{S}(\mu^*)$. This argument does not hold if the multiplier vector μ^* is changed between nodes.

Separation Rules—Separation of the solution space is accomplished by choosing a variable from among the free variables and fixing this variable to zero and one respectively. Two rules were investigated for selecting the separation variable:

(1) Ratio Rule. Order the variables in decreasing order of $c_i/(\mu \cdot A)_i$ and use this order as the separation order. Note that the order is redefined every time a new set of multipliers are generated.

(2) Penalty Rule. Use penalties of δ_{i_2} from the sensitivity analysis and order the variables according to the decreasing order of penalties. This order is used as the separation order. Note that generation of a new set of multipliers is not sufficient to redefine the separation order. In order to do so, sensitivity analysis is required.

Branching Rules—After a variable is selected and two nodes are generated according to the separation rule, one of these nodes is chosen using the branching rule. The branching rule also dictates the selection of a node among all pending nodes after a node is fathomed. Two branching rules, each one corresponding to one of the separation rules, are considered here. These rules differ in the branching action after the separation step. Rule 1 which is used with separation rule 1, always selects the node that is generated by fixing the separation variable x_i to 1. On the other hand, rule 2 picks the node that is generated by fixing x_1 to x_i^*; x_i^* is the value that x_i assumed in the solution to $S(\mu^*)$ which was the basis of the sensitivity analysis determining the current separation order. Both rules converge to a depth first rule with respect to the branching action taken after fathoming a node. With a depth first rule the algorithm backtracks to the first pending node and a new forward path is started from that point. This rule guarantees a linear storage space requirement during the implementation of the algorithm.

Sensitivity Analysis—Sensitivity analysis and reduction tests can be carried out at various levels of the branch and bound tree to fix some of the free variables and to define new penalties which may then be used for updating the order of separation in the second rule. Sensitivity analysis is potentially most powerful after reoptimizing the multiplier vector, even though it is possible to perform this analysis at other nodes of the tree.

An experimental branch and bound code capable of implementing the bounding, separation and branching rules was developed. Experiments were carried out to test the effectiveness of different strategies formed by combining various rules. It was previously observed that a tradeoff can be made between the time spent for reoptimizing multipliers and the time spent for searching a larger branching tree. At one extreme, the multipliers determined at the root node can be used without any reoptimization. The other extreme is to reoptimize at every node and obtain the smallest tree possible for fixed branching and separation rules. Other strategies can be generated by varying the reoptimization frequency of the multipliers; those will fall between those two extremes.

Three sets of experiments were carried out to compare various bounding, branching and separation rules. In each set of experiments, bounding and sensitivity analysis schemes were kept fixed and the separation and branching rules were varied. Six sets of problems, each containing ten problems were generated. Those were a subset of the problems used in previous sections. The experiments considered three methods for calculating the multipliers.

Method 1—The multipliers were computed once and were not updated during the branch and bound process.

Method 2—Here the multipliers were recomputed in every node with potential for bound improvement as previously explained under 'Bounding Rules', and sensitivity analysis was done after each reoptimization.

Method 3—Is similar to Method 2, except that reoptimization is not done at lower levels of the branching tree. In this method, reoptimization is done only if the total

cost associated with variables already fixed to one is less than 85 percent of the best known feasible solution.
A detailed description of those experiments and the results can be found in [15]. It was observed that Method 2 produced the lowest number of nodes in the branching tree and Method 1 the highest. When the total number of 0–1 knapsack problems solved is compared, Method 3 requires the solution of a significantly smaller number of knapsack problems than Method 2.

5. Performance of the algorithm

Based on the computational experience with the experimental code, a working code has been developed. It uses the first separation rule coupled with the bounding scheme used in reoptimization Method 1. This rule lends itself to an efficient implementation. Since the order of separation is the same as the order used for solving $\bar{S}(\mu)$, it is very easy to calculate the bound. In the remainder of the paper, this algorithm will be referred to as Procedure-GP.

5.1. Performance of Procedure-GP

Procedure-GP was coded in FORTRAN and tested on 15 groups of problems, consisting of 10 problems each. The times for each phase of the algorithm as well as total solution times are reported in Table 6. All times are in IBM 3032 CPU seconds. As Table 6 indicates, the algorithm is capable of solving fairly large problems in reasonable computing times. Problems with 3 constraints are easily solved regardless of the number of variables. The problems with 5 constraints are harder and take noticeably more time to solve. For problems with 200 variables and 5 constraints, an optimal solution was not found after 90 seconds in 4 out of 30 cases. But even in those cases, the best feasible solution at termination was rather close to the upper bound indicating that these solutions are either optimal or are very close to it.

5.2. The impact of changes in problem data

Procedure-GP was further tested on different groups of problems in order to identify the capabilities of the algorithm. The first set was designed to test the effect of tightening the right-hand side vector while keeping A and c constant. The division factor α was changed in the formula $b_i = \sum_j a_{ij}/\alpha$ generating different problems. For each group 10 problems were generated. The results are reported in Table 7. By comparing the first row of the table with the rest of the rows, it is possible to conclude that the problems are most difficult when the division factor is 2. A partial explanation for this phenomenon is the way the problems are generated. If a list of variables is constructed in decreasing order of ' attractiveness ' then there will

Table 6

Solution times with Procedure-GP, ($A \sim U(1, 100)$, $c \sim U(1, 100)$)

Right hand side element (b_i)	Number of constr. (m)	Number of variables (n)	Time spent at the root of the branch and bound tree			Time in branch and bound			Total time (sec)		
			Min.	Mean	Max.	Min.	Mean	Max.	Min.	Mean	Max.
$\sum_j a_{ij}/2$	3	100	0.33	0.47	0.53	0.03	0.15	0.53	0.33	0.62	1.02
		200	0.90	1.19	1.49	0.06	0.33	0.80	0.90	1.52	2.30
		300	1.86	2.31	2.75	0.26	1.38	2.60	2.25	3.69	5.04
$\sum_j a_{ij}/4$	3	100	0.36	0.45	0.53	0.03	0.12	0.27	0.39	0.57	0.77
		200	1.09	1.35	1.64	0.14	1.44	4.05	1.13	2.79	5.06
		300	2.00	2.60	3.05	0.63	3.56	11.48	3.12	6.16	13.65
$\sum_j a_{ij}/4 - \sum_j a_{ij}/2$	3	100	0.32	0.48	0.75	0.02	0.15	2.01	0.26	0.63	1.76
		200	0.78	1.19	1.97	0.02	0.51	2.42	0.77	1.70	4.41
		300	1.67	2.25	3.35	0.09	0.71	3.15	1.79	2.96	6.50
$\sum_j a_{ij}/2$	5	100	0.54	0.66	0.71	0.14	1.61	3.90	0.65	2.27	4.65
		200[a]	0.15	1.16	2.40	1.40	15.86	>90	2.68	17.02	>90
$\sum_j a_{ij}/4$	5	100	0.54	0.74	1.04	0.14	3.05	11.40	0.63	3.79	12.36
		200[a]	1.55	2.04	3.17	10.71	35.30	90	12.21	37.34	90
$\sum_j a_{ij}/4 - \sum_j a_{ij}/2$	5	100	0.44	0.62	0.78	0.08	0.78	3.30	0.56	1.40	4.01
		200[a]	1.26	1.94	3.41	0.09	8.24	>90	1.83	10.18	>90

[a] Those groups have at least one problem that was terminated after 90 seconds. Averages are calculated excluding these problems.

Table 7
Effect of changing the right-hand side vector b ($n = 100$, $m = 5$, $A \sim U(1, 1000)$, $c \sim U(1, 1000)$)

Right hand side element (b_i)	Time in phase I			Time in branch and bound			Total time (sec)		
	Min.	Mean	Max.	Min.	Mean	Max.	Min.	Mean	Max.
$\sum a_{ij}/2$	0.45	0.52	0.60	0.30	3.79	16.90	0.75	4.31	17.50
$\sum a_{ij}/4$	0.39	0.63	1.16	0.36	1.80	5.46	0.74	2.43	6.61
$\sum a_{ij}/6$	0.41	0.62	0.79	0.06	2.06	8.42	0.47	2.68	8.99
$\sum a_{ij}/8$	0.41	0.51	0.61	0.20	2.26	6.79	0.66	2.77	7.26
$\sum a_{ij}/10$	0.41	0.59	0.77	0.16	1.28	5.19	0.73	1.87	5.72

be many ‘ almost equally attractive ’ variables around the middle of this list. For a division factor of 2, many of these variables will be packed in knapsacks making the problem more difficult since the algorithm has to choose from among "almost equally attractive" variables. As the capacity of knapsacks is decreased, these variables will be dominated by the more attractive variables and will be excluded from the solution, making the problem easier.

The second experiment was generated by varying the range of the uniform distribution from which the elements of A were drawn. The right-hand side vector elements were determined as in experiment 1 with a division factor of 2. The results are reported in Table 8. As the range of A is increased from 10 to 1000, the problems get harder. There is a slight decrease in difficulty when this range is increased to 5000 from 1000.

In the third experiment, the range of the uniform distribution for generating c was varied. The results reported in Table 9 point to the fact that perhaps the problems tend to be more difficult when c is generated from a small interval, e.g., between 1 and 10.

Table 8
Effect of changing the range of the uniform distribution for generating A ($n = 100$; $m = 5$; $b_i = \sum_j a_{ij}/2$; $c \sim U(1, 1000)$)

Elements of A are drawn from	Time in phase I			Time in branch and bound			Total time (sec)		
	Min.	Mean	Max.	Min.	Mean	Max.	Min.	Mean	Max.
$U(1, 10)$	0.40	0.55	0.72	1.00	0.66	1.40	0.54	1.21	2.02
$U(1, 50)$	0.47	0.63	1.01	0.22	1.99	7.08	1.07	2.62	8.09
$U(1, 100)$	0.44	0.54	0.71	0.24	3.43	16.47	0.69	3.97	17.18
$U(1, 1000)$	0.42	0.53	0.62	0.32	4.12	18.36	0.77	4.65	18.97
$U(1, 5000)$	0.41	0.54	0.67	0.42	3.84	17.46	1.09	4.38	17.98

Table 9

Effect of varying the range of the uniform distribution for generating c ($n = 100$; $m = 5$; $b_i = \sum_j a_{ij}/2$; $A \sim U(1, 1000)$)

Elements of c are drawn from	Time in phase I			Time in branch and bound			Total time (sec)		
	Min.	Mean	Max.	Min.	Mean	Max.	Min.	Mean	Max.
$U(1, 10)$	0.84	1.60	4.11	0.54	6.48	21.23	1.39	8.08	22.64
$U(1, 50)$	0.55	0.82	1.55	0.60	3.86	14.60	1.29	4.68	15.68
$U(1, 100)$	0.66	0.85	1.26	0.39	4.00	15.86	1.19	4.85	17.11
$U(1, 1000)$	0.42	0.53	0.62	0.32	4.11	18.36	0.77	4.64	18.97
$U(1, 5000)$	0.43	0.59	0.79	0.41	4.05	17.03	0.91	4.64	17.72

A set of difficult problems was generated by correlating the elements of c with the corresponding columns of A. Elements of c were defined using the following formula:

$$c_j = \sum_i a_{ij}/m + R * K$$

where m is the number of constraints, R is a number drawn from $U(0, 1)$ and K is a constant. The objective function vector c defined this way is not only correlated to the coefficient matrix A but also consists of elements which lie in a small range around the mean value of the uniform distribution from which A is generated. An experiment was carried out where elements of A are drawn from $U(1, 1000)$ and the right-hand side vector was determined as in the previous experiment. The constant K was varied between 0 and 500 and the results are reported in Table 10. For small values of K, the problems are considerably more difficult than for large values. In fact, in each one of the groups reported in the first three rows of the table, there was one problem out of ten which was not solved optimally after 100 seconds in the branch and bound phase of Procedure-GP. This observation suggests that the problems which have A and c correlated tend to be more difficult than those with no correlation.

Table 10

Effect of correlating c to A ($n = 100$, $m = 5$, $b_i = \sum_j a_{ij}/2$, $A \sim U(1, 000)$)

	Time in phase I			Time in branch and bound			Total time (sec)		
K	Min.	Mean	Max.	Min.	Mean	Max.	Min.	Mean	Max.
0	0.78	1.07	1.50	0.24	27.67	100.00	1.02	28.74	101.00
10	0.75	1.03	1.28	0.54	29.24	100.00	1.81	30.27	101.00
100	0.73	0.98	1.66	2.00	20.23	100.00	3.28	21.21	100.80
500	0.45	0.61	0.76	0.76	4.49	16.13	1.21	5.10	16.88

5.3. Comparison of Procedure-GP with other codes

Procedure-GP was compared to two recently developed codes. The first one of these was a special purpose code developed by Shih [38] for solving multiconstraint knapsack problems. In the following discussion this code will be referred to as Procedure-S. The second code was a state of the art commercial integer programming package known as Sciconic/VM [40].

The original implementation of Procedure-S, supplied by Shih, was used in the computational experiments. Shih's code requires large amounts of memory. In the following tests, it was provided with two MBytes of memory for storing its search tree. Sixteen sets of twenty problems each (320 problems in total) were generated. Table 11 summarizes the data generation method and computational results for each set of problems. Computing times are given in CPU seconds of an AMDAHL 470-V8.

Procedure-GP has solved without any difficulty, all the problems that were generated. Procedure-S was able to solve relatively small problems ($n = 20$) or problems with a few nonbinding constraints. For tight resource constraints and $n \geq 60$, Shih's procedure was not able to solve even a single problem before the memory which was allocated to it had been exhausted. Even for the problems that

Table 11
Comparison of Procedure-S and Procedure-GP

Data generation method			AMDAHL 470-V8 CPU Time (seconds)					
			Procedure-S			Procedure-GP		
a_{ij}, c_j, b_i	m	n	Min.	Mean	Max.	Min.	Mean	Max.
$a_{ij} \sim U(30, 100)$	3	20	0.275	0.547	1.333	0.006	0.030	0.070
$c_j \sim U(50, 120)$	3	40	0.817	1.287	2.704	0.008	0.029	0.084
$b_i \sim U(30m, 60m)$	3	60	1.108	20.606[a]		0.011	0.112	0.394
	3	80	1.347	3.635	4.843	0.017	0.039	0.109
	5	20	0.318	0.692	1.208	0.005	0.028	0.071
	5	40	1.405	25.286[a]		0.036	0.087	0.146
	5	60	1.191	39.404[a]		0.016	0.150	0.236
	5	80	4.282	42.721[a]		0.024	0.315	0.577
$a_{ij} \sim U(1, 100)$	3	20	0.304	0.399		0.021	0.031	0.051
$c_j \sim U(1, 100)$	3	40	9.717	22.529[a]		0.067	0.078	0.111
$b_i = \sum a_{ij}/2$	3	60	46.222[a]			0.119	0.189	0.260
	3	80	51.967[a]			0.299	0.478	0.933
$a_{ij} \sim U(1, 100)$	3	20	0.325	0.546	0.884	0.019	0.025	0.030
$c_j \sim U(1, 100)$	3	40	23.256	34.198[a]		0.073	0.111	0.176
$b_i = \sum a_{ij}/4$	3	60	50.446[a]			0.127	0.219	0.302
	3	80	52.446[a]			0.267	0.538	0.732

[a] Some problems were terminated before completion of search.

Table 12

Comparison of Sciconic/VM and Procedure-GP

Data generation method			CPU Seconds (PRIME 550 Mod. II)						Mean (SC)/ Mean (GP)
			Procedure-GP			Sciconic/VM			
a_{ij}, c_j, b_i	m	n	Min.	Mean (GP)	Max.	Min.	Mean (SC)	Max.	
$a_{ij} \sim U(1, 100)$	3	40	1.2	1.8	2.9	30.6	78.2	152.1	44.9
$c_j \sim U(1, 100)$	5	60	1.8	2.9	3.8	87.3	162.1	240.5	55.9
$b_i = \sum_j a_{ij}/2$	7	80	4.6	5.5	7.9	338.1	479.2	744.3	87.1
		40	2.0	3.6	7.3	70.5	108.6	138.4	30.2
		60	2.1	4.4	9.1	65.9	231.2	629.7	52.6
		80	5.5	11.2	17.2	78.2	705.1	1697.8	63.0
		40	2.7	4.3	6.2	68.9	144.2	279.5	33.5
		60	4.8	14.5	34.9	89.8	593.2	1376.3	40.9
		80	7.2	24.2	63.0	335.2	1006.2	2961.8	41.5
$a_{ij} \sim U(1, 100)$	3	40	1.3	1.7	3.1	62.9	125.2	197.2	73.6
$c_j \sim U(1, 100)$	5	60	2.7	3.7	5.1	98.6	251.2	476.5	67.9
$b_i = \sum_j a_{ij}/4$	7	80	4.8	7.8	11.4	262.4	622.2	971.3	79.8
		40	2.6	3.7	4.2	56.2	127.8	237.4	34.5
		60	3.8	8.0	13.8	118.3	326.6	658.2	40.8
		80	10.6	24.3	46.8	490.8	1160.8	1715.8	47.8
		40	2.1	8.6	17.7	124.5	190.2	250.2	22.1
		60	6.0	50.3	109.9	493.8	1128.6	1986.7	22.5
		80	17.7	74.5	159.0	1374.2	2342.6	3321.1	31.4

were solved to optimality, the new algorithm exhibits at least one order of magnitude improvement in CPU time versus Procedure-S.

Procedure-GP and Sciconic/VM were compared on 18 problem sets each containing 20 problems randomly generated in the same manner. Data generation methods as well as computational results are summarized in Table 12. Experiments were performed on a PRIME 550 Mod. II running under PRIMOS Rev. 19.2.7. and the computational times are reported in CPU seconds of this machine.

On all 360 problems solved, Procedure-GP was significantly faster than Sciconic/VM. Minimum, mean and maximum computing times with each algorithm are reported for every problem set. Also reported are the ratios obtained from the division of mean computation times with Sciconic/VM by mean computation times with Procedure-GP. The ratio suggest that for fixed m (number of constraints) Procedure-GP becomes comparatively more efficient as the number of variables (n) is increased. On the other hand for fixed n the relative efficiency of Procedure/GP declines as m is increased.

5.4. *Using Procedure-GP as a heuristic*

From the previous experiments, it is evident that solution times grow significantly with the increase in problem size. In many cases, especially in large-scale problems, it is possible to reduce computing times for an approximate solution which is close enough to the optimal solution. In such cases, Procedure-GP can be used as a heuristic. It was observed that the procedure identifies early in the branch and bound search a feasible solution which is optimal or very close to optimality. In such cases, it is possible to terminate the algorithm before verifying that the solution is optimal.

Table 13 illustrates the impact of early termination of Procedure-GP. In all cases, the algorithm was terminated 0.15 seconds after completing the generation of the root node of the branching tree. The best feasible solution generated by termination was printed as output from the heuristic. In all the 240 cases tested, the early termination heuristic generated solutions within half a percent from the optimum. In many instances, the solution found by the heuristic was in fact the optimal solution. CPU times for early termination were significantly shorter than the CPU times needed for a complete search, especially in problems that require extensive computing times.

In order to demonstrate the effectiveness of this procedure, it was compared to Heuristic-SW1 in Loulou and Michaelides [33] which is considered to be the best heuristic available in terms of solution quality. The solution times with the early termination procedure are higher than the times with the Heuristic-SW1, but the results reported in Table 13 clearly demonstrate that the early termination heuristic generates solutions that are significantly closer to optimality. Heuristic-SW1 has generated the optimal solution in only one out of 240 problems tested, while the early termination heuristic in 85 problems.

Summary

We have developed and compared various relaxations of the multiconstraint 0–1 knapsack problem. New algorithms for deriving surrogate multipliers were proposed and reduction procedures for the multiconstraint 0–1 knapsack problem were introduced and tested. Experiments with a general branch and bound algorithm have been performed in order to determine the effectiveness of different branching, bounding and separation schemes. The final algorithm was compared with an optimal algorithm developed by Shih [36] and was found to be faster by at least one order of magnitude. It was also compared to and found to be significantly faster than Sciconic/VM [40], a state of the art commercial integer programming algorithm. It was suggested that the algorithm can be used as a heuristic by terminating it before the tree search is completed. The effectiveness of this scheme was computationally tested and found to be superior to a heuristic developed and tested by Loulou and Michaelides [33]. A copy of the final code can be obtained upon request from one of the authors.

Table 13

Performance of the Early Termination Heuristic and Heuristic-SW1 ($A \sim U(1, 1000)$, $c \sim U(1, 1000)$, CPU times on an AMDAHL 470-V8)

Data generation method			Performance of early termination heuristic							Mean CPU time for complete *B & B* search	Performance of LouLu and Michaelides heuristic					
			Gap to optimal solution (%)				CPU time (sec)				Gap to optimal solution (%)			CPU time (sec)		
b_i	m	n	Min.	Mean	Max.	Optimal[a]	Min.	Mean	Max.	(Sec)	Min.	Mean	Max.	Min.	Mean	Max.
$\sum a_{ij}/2$	3	100	0	0.00	0.00	100	0.307	0.439	0.705	0.667	0.00	1.05	1.96	0.080	0.086	0.087
		200	0	0.02	0.05	50	1.117	1.411	1.772	2.030	0.86	1.55	2.60	0.321	0.333	0.342
		300	0	0.03	0.08	30	2.704	3.293	4.493	5.899	1.17	1.75	2.42	0.740	0.748	0.759
		400	0	0.03	0.05	30	3.720	7.020	12.170	17.780	1.22	1.83	2.74	1.130	1.150	1.170
		500	0	0.02	0.10	0	5.740	7.360	9.960	67.340	1.48	1.87	2.70	1.760	1.790	1.810
	5	100	0	0.04	0.15	50	0.406	0.559	0.681	2.191	0.51	1.16	2.05	0.115	0.118	0.122
		200	0	0.09	0.28	20	1.319	1.768	2.277	11.482	0.45	1.16	1.94	0.460	0.469	0.476
		300	0	0.06	0.12	10	3.410	4.080	4.850	50.690	0.62	1.14	1.67	0.890	0.920	0.930
$\sum a_{ij}/4$	3	100	0	0.16	1.39	70	0.222	0.356	0.586	0.653	3.14	4.91	7.00	0.049	0.051	0.053
		200	0	0.08	0.23	10	0.944	1.379	1.984	5.310	1.55	5.44	9.73	0.188	0.199	0.211
		300	0	0.06	0.12	20	1.989	3.363	5.008	9.228	4.17	5.64	7.56	0.434	0.451	0.475
		400	0	0.06	0.10	10	4.150	5.430	7.630	13.960	4.27	6.19	7.87	0.650	0.670	0.710
		500	0	0.04	0.10	10	7.520	9.730	12.200	26.680	4.87	6.59	8.08	1.030	1.060	1.090
	5	100	0	0.34	1.24	30	0.399	0.552	0.718	6.277	3.01	4.87	9.83	0.065	0.070	0.076
		200	0	0.08	0.20	30	1.373	1.837	2.666	31.035	1.77	4.84	7.20	0.266	0.277	0.284
		300	0	0.17	0.41	10	2.710	3.660	4.660	45.060	3.57	4.52	5.82	0.520	0.540	0.560
$\sum a_{ij}/4 -$	3	100	0	0.04	0.41	90	0.093	0.270	0.455	0.471	1.01	2.92	5.48	0.058	0.066	0.076
$\sum a_{ij}/2$		200	0	0.04	0.09	40	0.890	1.369	1.810	1.790	3.02	3.84	4.59	0.243	0.265	0.297
		300	0	0.01	0.05	40	0.076	2.353	3.252	4.465	1.97	4.96	8.18	0.517	0.588	0.667
		400	0	0.03	0.07	10	4.410	5.800	8.970	9.640	1.52	4.25	7.55	0.720	0.900	1.100
		500	0	0.01	0.03	20	0.130	8.710	14.410	39.210	2.88	6.67	9.90	1.120	1.320	1.600
	5	100	0	0.01	0.10	90	0.172	0.414	0.609	1.665	1.45	2.84	5.39	0.079	0.093	0.105
		200	0	0.05	0.15	40	0.882	1.302	1.780	3.593	2.81	4.12	6.09	0.306	0.349	0.395
		300	0	0.04	0.23	40	2.440	3.310	4.840	9.580	2.24	4.40	7.01	0.570	0.680	0.790

[a] Percent of cases in which the heuristic generated the optimal solution.

Acknowledgements

We are indebted to Professor W. Shih for providing us with a listing of his program, and to the referees for their valuable comments on an earlier version of the paper.

References

[1] E. Balas and E. Zemel, "An algorithm for large zero–one knapsack problems", *Operations Research* 28 (1980) 1130–1154.

[2] E. Balas and G.H. Martin, "Pivot and complement – A heuristic for 0–1 programming", *Management Science* 26 (1980) 86–96.

[3] R.L. Bulfin, P.G. Parker and C.M. Shetty, "Computational results with a branch and bound algorithm for the general knapsack problem", *Naval Research Logistics Quarterly* 26 (1979) 41–46.

[4] A.V. Cabot, "An enumeration algorithm for knapsack problems", *Operations Research* 18 (1970) 306–311.

[5] L.G. Chalmet and L.F. Gelders, "Lagrangian relaxations for a generalized assignment-type problem", Proceedings of Second European Congress on Operations Research (North-Holland, 1976) pp. 103–109.

[6] M.E. Dyer, "Calculating surrogate constraints", *Mathematical Programming* 19 (1980) 255–278.

[7] D. Erlenkotter, "A dual based procedure for uncapacitated facility location", *Operations Research* 26 (1978) 992–1009.

[8] H. Everett, "Generalized lagrange multiplier method for solving problems of optimum allocation of resources", *Operations Research* 11 (1963) 399–417.

[9] B. Gavish, "On obtaining the 'best' multipliers for a lagrangian relaxation for integer programming", *Computers and Operations Research* 5 (1978) 55–71.

[10] B. Gavish, "Topological design of centralized computer networks – Formulations and algorithms", *Networks* 12 (1982) 355–377.

[11] B. Gavish, "Formulations and algorithms for the capacitated minimal directed tree problem", *Journal of the Association for Computing Machinery* 30 (1983) 118–132.

[12] B. Gavish and S.L. Hantler, "An algorithm for optimal route selection in SNA networks", *IEEE Transactions on Communications* 31 (1983) 1154–1161.

[13] B. Gavish and H. Pirkul, "Allocation of data bases and processors in a distributed computing system" in: J. Akoka, ed., *Management of distributed data processing* (North-Holland, 1982) pp. 215–231.

[14] B. Gavish and H. Pirkul, "Models for computer and file allocation in distributed computer networks", Working Paper, The Graduate School of Management, The University of Rochester (Rochester, 1983).

[15] B. Gavish and H. Pirkul, "The multiconstraint 0–1 knapsack problem", Working Paper, The Graduate School of Management, The University of Rochester (Rochester, 1981).

[16] B. Gavish and K.N. Srikanth, "An optimal solution method for the multiple travelling salesman problem", Working Paper, The Graduate School of Management, The University of Rochester (Rochester, 1980).

[17] A.M. Geoffrion, "Lagrangian relaxation and its uses in integer programming", *Mathematical Programming Study* 2 (1974) 82–114.

[18] A.M. Geoffrion and R. McBride, "Lagrangian relaxation applied to capacitated facility location problems", *AIIE Transactions* 10 (1978) 40–47.

[19] P.C. Gilmore and R.E. Gomory, "Theory and computation of knapsack problems", *Operations Research* 14 (1966) 1045–1074.

[20] F. Glover, "A multiphase dual algorithm for the zero–one integer programming problem", *Operations Research* 13 (1965) 879–919.

[21] F. Glover, "Surrogate constraint duality in mathematical programming", *Operations Research* 23 (1975) 434–453.

[22] C.J. Green, "Two algorithms for solving independent multidimensional knapsack problems", Management Science Research Report No. 110, Carnegie Institute of Technology, Graduate School of Industrial Administration (Pittsburgh, 1967).

[23] H.J. Greenberg, "The generalized penalty-function/surrogate model", *Operations Research* 21 (1973) 162–178.

[24] H.J. Greenberg and W.P. Pierskalla, "Surrogate mathematical programming", *Operations Research* 18 (1970) 924–939.

[25] M. Held and R.M. Karp, "The travelling salesman problem and minimum spanning trees", *Operations Research* 18 (1970) 1138–1162.

[26] M. Held, P. Wolfe and H.P. Crowder, "Validation of subgradient optimization", *Mathematical Programming* 5 (1974) 62–88.

[27] E. Horowitz and S. Sahni, "Computing partitions with applications to the knapsack problem", *Journal of the Association for Computing Machinery* 21 (1974) 277–292.

[28] G.P. Ingargiola and J.F. Korsh, "Reduction algorithm for zero–one single knapsack problems", *Management Science* 20 (1973) 460–463.

[29] M.H. Karwan, "Surrogate constraint duality and extensions in integer programming", Ph.D. Dissertation, School of Industrial and Systems Engineering, Georgia Institute of Technology (Atlanta, 1976).

[30] M.H. Karwan and R.L. Rardin, "Some relationships between lagrangian and surrogate duality in integer programming", *Mathematical Programming* 18 (1979) 320-334.

[31] M.H. Karwan and R.L. Rardin, "Searchability of composite and multiple surrogate dual functions", Working Paper, School of Industrial and Systems Engineering, Georgia Institute of Technology (Atlanta, 1978).

[32] A.H. Land and S. Powell, "Fortran Codes for mathematical programming linear, quadratic and discrete", Wiley (London, 1973).

[33] R. Loulou and E. Michaelides, "New greedy-like heuristics for the multidimensional 0–1 knapsack problem", *Operations Research* 27 (1979) 1101–1114.

[34] D.G. Luenberger, "Quasi-convex programming", *Siam Journal of Applied Mathematics* 16 (1968) 1090–1095.

[35] S. Martello and P. Toth, "An upperbound for the zero–one knapsack problem and a branch and bound algorithm", *European Journal of Operational Research* 1 (1977) 168–175.

[36] R.M. Nauss, "An efficient algorithm for the 0–1 knapsack problem", *Management Science* 23 (1976) 27–31.

[37] H.M. Salkin and C.A. Kluyver, "The knapsack problem, a survey", *Naval Research Logistics Quarterly* 22 (1975) 127–144.

[38] W. Shih, "A branch and bound method for the multiconstraint zero–one knapsack problem", *Journal of Operational Research Society* 30 (1979) 369–378.

[39] A.L. Soyster, B. Lev and W. Slivka, "Zero–one programming with many variables and few constraints", *European Journal of Operational Research* 2 (1978) 195–201.

[40] Users Guide to Scionic/VM (Version 1–2), Scicon Computer Services, Ltd., Brick Close, Kiln Farm, Milton Keynes MK11 3EJ, U.K.

[41] H.M. Weingartner and D.N. Ness, "Methods for the solution of the multidimensional 0–1 knapsack problem", *Operations Research* 15 (1967) 83–103.