A HYBRID ALGORITHM FOR THE MULTIPLE CHOICE KNAPSACK

James C. Bean
and
Kirsten A. Syverson
Department of
Industrial & Operations Engineering
University of Michigan
Ann Arbor, MI 48109-2117

Technical Report 90-33

November 1990

A HYBRID ALGORITHM FOR THE MULTIPLE CHOICE KNAPSACK

James C. Bean and Kirsten A. Syverson
Department of Industrial and Operations Engineering
The University of Michigan
Ann Arbor, MI 48109-2117

November 16, 1990

Abstract

We present a hybrid algorithm based on state and variable elimination in a dynamic program for the multiple choice knapsack problem. The algorithm uses linear multiple choice knapsack solutions to provide bounds on the optimal value of a state in the dynamic program. The algorithm is competitive with the best published algorithms in run times and is simpler to code.

OR/MS Keywords: 069 Dynamic programming, deterministic; 712 Programming, integer, algorithms

The multiple choice knapsack is a knapsack problem with exhaustive multiple choice constraints, and is formulated:

$$\max \sum_{i=1}^{M} \sum_{j=1}^{n_i} c_{ij} x_{ij}$$

$$\text{subject to : } \sum_{i=1}^{M} \sum_{j=1}^{n_i} a_{ij} x_{ij} \leq B$$

$$\sum_{j=1}^{n_i} x_{ij} = 1, \quad i = 1, 2, \ldots, M \qquad (MK_{(M,B)})$$

$$x_{ij} \in \{0, 1\}, \quad i = 1, 2, \ldots, M;\ j = 1, 2, \ldots, n_i.$$

The formulation requires that exactly one element of each multiple choice be set equal one, while not exceeding a knapsack capacity, B. Without loss of generality, we may assume that $c_{ij} \geq 0$. If not, add a sufficient constant to the objective coefficient of each variable in the corresponding multiple choice set. Also, without loss of generality, assume that c_{ki} and a_{ki} are strictly monotonically increasing in i. It is well known that variables can be

eliminated *a priori* if $c_{ki} \geq c_{kj}$ and $a_{ki} \leq a_{kj}$. The linear multiple choice knapsack problem is the linear programming relaxation of $MK_{(M,B)}$ and is denoted $LMK_{(M,B)}$.

The literature mentions several applications of the multiple choice knapsack such as capital budgeting and menu planning (Sinha and Zoltners [1979]). We were motivated to consider this problem during a project on shopping mall merchandising with Homart Development Co. (see Bean, et al. [1987]). In this problem we seek to maximize the profit from a shopping mall by selecting an appropriate set of stores for the hallways between the department stores. A set of potential stores is considered, each having a small number of possible sizes. In the formulation above, i runs through the stores and j enumerates the sizes for store i. The knapsack constraint forces us within the square footage limit of the mall.

The $MK_{(M,B)}$ and its variations have been studied frequently in the literature, e.g., Dyer, et al. [1984], Dudzinski [1984], Zemel [1980], Sinha and Zoltners, Nauss [1975] and Lesso [1970]. The traditional solution approach has been branch and bound supported by linear programming relaxations. This technique is attractive since $LMK_{(M,B)}$ has special structural properties allowing $O(n)$ solution. As with all branch-and-bound schemes, these approaches have relatively high variance in computation times and can be tedious to implement.

Lesso and, independently, Dudzinski formulated $MK_{(M,B)}$ as a dynamic program. They develop a multiple choice knapsack function parallel to the knapsack function in Gilmore and Gomory [1966]. Such approaches are easy to implement and have predictable computation times, though neither author discusses implementation. However, computation times for knapsack type functions degrade as B increases and, as dual algorithms, do not have a feasible solution until completion.

This paper presents a hybrid algorithm, similar to that introduced by Morin and Marsten [1976], based on the dynamic programming approach. Hybrid algorithms use characteristics of both branch and bound and dynamic programming to accelerate computation. Linear relaxations of the integrality constraints of $MK_{(M,B)}$ are used to provide bounds on the optimal solution and eliminate nodes from the dynamic programming state

space. The hybrid, unlike pure dynamic programming, is a primal method. A feasible solution is maintained throughout computation. With these improvements, we present the first implemented, computationally tractable dynamic programming algorithm for the multiple choice knapsack problem. The hybrid algorithm is computationally competitive with the best branch-and-bound codes and simpler to code.

In Section 1 we present a dynamic programming approach. Section 2 develops the hybrid algorithm. Section 3 contains computational results. Section 4 shows application of the techniques to real problems from Homart mall merchandising. Section 5 is a summary and conclusions.

1. Dynamic Programming Formulation

In $MK_{(M,B)}$ let $N = \sum_{i=1}^{M} n_i$ be the number of variables. Rather than viewing the problem as making N binary decisions, view each multiple choice set as one decision, that is, which element to choose from the set. This view has been used successfully for the multiple choice integer program (Bean [1984]).

Formulate a dynamic program with states consisting of ordered pairs, (k, β), where $0 \leq k \leq M$, $0 \leq \beta \leq B$. The multiple choice knapsack function, $f(k, \beta)$, is the optimal value for $MK_{(k,\beta)}$. That is, $f(k, \beta)$ equals

$$\max \sum_{i=1}^{k} \sum_{j=1}^{n_i} c_{ij} x_{ij}$$

$$\text{subject to}: \quad \sum_{i=1}^{k} \sum_{j=1}^{n_i} a_{ij} x_{ij} \leq \beta$$

$$\sum_{j=1}^{n_i} x_{ij} = 1, \quad i = 1, 2, \ldots, k \qquad (MK_{(k,\beta)})$$

$$x_{ij} \in \{0, 1\}, \quad i = 1, 2, \ldots, k; \; j = 1, 2, \ldots, n_i.$$

Thus, $f(M, B)$ is the optimal value to $MK_{(M,B)}$. We recursively solve a collection of these problems to determine $f(M, B)$.

From a state (k, β), the residual problem is the multiple choice knapsack of size $B - \beta$ considering sets $k+1$ through M. Define $f'(k+1, B-\beta)$ as the optimal value to this problem. Lesso and Dudzinski imply, though do not state, the following Theorem.

Theorem 1: If an optimal solution to $MK_{(M,B)}$ has a subcollection of items from multiple choice sets 1 through k having aggregate size β, this subcollection must be optimal for $MK_{(k,\beta)}$, that is, $f(M,B) = f(k,\beta) + f'(k+1, B-\beta)$.

Theorem 1 implies that the optimal value, $f(k,\beta)$, can be found from the solution to $f(k-1,b)$ for $b < \beta$. The optimal value, $f(M,B)$, is found by solving forward with the recursive equations:

Boundary Condition:

$$f(1,\beta) = \max\{\max_{1 \le j \le n_1} (c_{1j} : a_{1j} \le \beta), 0\}, \beta = 1,2,\ldots,B; \quad f(k,0) = 0, \text{ all } k \qquad (1)$$

Functional Equation:

For $k = 2$ to M solve

$$f(k,\beta) = \max_{1 \le j \le n_k} \{f(k-1, \beta - a_{kj}) + c_{kj} : a_{kj} \le \beta, f(k-1, \beta - a_{kj}) > 0\}, \qquad (2)$$

Solution:

$$f(M,B) = \max_{\beta \le B}\{f(M,\beta)\}, \qquad (3)$$

The dynamic programming network implied by these functional equations has arcs from state $(k-1, \beta - a_{ij})$ to (k,β), with value c_{ij}, for $j = 1$ to n_k. If no combination of items from the first k sets has size exactly β, then $f(k,\beta) = 0$. Lesso and Dudzinski solved this recursion with a recursive fixing algorithm. Below we define a reaching algorithm for the same network (see Denardo [1982] for a discussion of fixing versus reaching). For the recursion stated above, fixing and reaching result in exactly the same computations. However, reaching allows pruning of nodes and arcs as described in Section 2. The following algorithm solves equations (1) to (3) by reaching.

Algorithm 1: (Dynamic Programming Algorithm)

Step 0: (Initialize) Set

$$f(1,\beta) = \max\{\max_{1\le j\le n_j}(c_{1j} : a_{1j} = \beta), 0\},$$

for $\beta = 1,2,\ldots,B$. Set $\beta = 1$, $k = 1$. Go to Step 1.

Step 1: (Reach from (k,β)) For $j = 1,2,\ldots,n_{k+1}$, if $\beta + a_{kj} \le B$, $f(k,\beta) > 0$ and $f(k,\beta) + c_{kj} > f(k+1,\beta+a_{kj})$, then let $f(k+1,\beta+a_{kj}) = f(k,\beta) + c_{kj}$ and $soln(k+1,\beta+a_{kj}) = j$. Let $\beta = \beta + 1$. If $\beta < B$ then go to Step 1. Else, go to Step 2.

Step 2: Let $k = k+1$ and $\beta = 1$. If $k < M$, then go to Step 1. Else, go to Step 3.

Step 3: (Find optimal solution) Let optimal value equal $\max_{1\le\beta\le B} f(M,\beta)$. Let b be the argument of this maximum. Let $k = M$ and $\beta = b$. Go to Step 4.

Step 4: (Recover solution) Let $x_{k,soln(k,\beta)} = 1, \beta = \beta - a_{k,soln(k,\beta)}, k = k-1$. If $k > 0$ then go to Step 4. Else, stop.

Computation in such algorithms increases in B. This effect is limited by the fact that if $B > \sum_{i=1}^{m} \max_j a_{ij}$ then the problem is trivial.

To enhance computation, we would like to eliminate nodes before reaching from them. The following Theorem allows elimination of some nodes. It is a variant of the "further but cheaper" pruning criteria used frequently in capacity expansion and equipment replacement (see Bean and Smith [1985]).

Theorem 2: If $\beta_1 \le \beta_2$ and $f(k,\beta_1) \ge f(k,\beta_2)$ then not all optimal solutions include the subset of items in $f(k,\beta_2)$. We say that $f(k,\beta_1)$ dominates $f(k,\beta_2)$.

Proof: If there does exist an optimal solution that includes the items corresponding to $f(k,\beta_2)$, then replace the knapsack of weight β_2 with that of weight β_1. By assumption, the knapsack will have higher value, at a lower weight (neither strict). By Theorem 1, their exchange will not necessitate a change in the solution to the residual problem. Therefore $f(k,\beta_2)$ can be replaced by $f(k,\beta_1)$ without loss of optimality. ∎

Theorem 2 implies that only states that constitute strict monotonic increases over the

previous states need be considered in the next stage of the algorithm. We can improve computational performance by modifying Step 1 of the algorithm so that reaching is only performed if $f(k,\beta) > f(k,\beta')$ for all $\beta' < \beta$.

2. Hybrid Algorithm

Hybrid algorithms were developed by Morin and Marsten [1976]. They combine bounding techniques common to branch-and-bound algorithms with a dynamic programming structure. The result is a computationally effective pruning technique to reduce the number of arcs and nodes that must be considered in the algorithm. Below we develop a hybrid technique for the multiple choice knapsack problem.

Define $LP_{(k+1,B-\beta)}$ to be the optimal value of the linear multiple choice knapsack of size $B-\beta$, including items from sets $k+1$ through M. That is, the linear counterpart to $f'(k+1,B-\beta)$. Then $LP_{(k+1,B-\beta)}$ is an upper bound on the optimal value of any completion of a state (k,β). Define LB to be the best known feasible solution to $MK_{(M,B)}$.

Theorem 3: If $f(k,\beta)+LP_{(k+1,B-\beta)} < LB$, then (k,β) cannot be in an optimal solution to $MK_{(M,B)}$.

Proof: We have

$$f(k,\beta)+f'(k+1,B-\beta) \leq f(k,\beta)+LP_{(k+1,B-\beta)} < LB \leq f(M,B).$$

Hence, by Theorem 1, (k,β) is not on an optimal path. ∎

To determine LB, Dantzig and Van Slyke [1965] show that the solution to $LP_{(k+1,B-\beta)}$ has at most two fractional variables, and that these variables come from the same multiple choice set. A lower bound on the optimal solution can be obtained by setting to one the fractional variable with smaller a_{ij}.

The hybrid algorithm uses the optimal linear completion from each state of the dynamic program to eliminate states which can be shown to have only suboptimal completions. Computational performance is enhanced by continually updating the value of LB. The algorithm requires solution of a sequence of related $LP_{(k+1,B-\beta)}$. We solve the

first such linear program, $LP_{(1,B)}$, by the method of Sinha and Zoltners. All subsequent solutions are simple updates.

Algorithm 2: (Hybrid Algorithm)

Step 0: (Initialize) Set

$$f(1,\beta) = \max\{\max_{1\le j\le n_j}(c_{1j} : a_{1j} = \beta), 0\},$$

for $\beta = 1, 2, \ldots, B$. Set $\beta = 1$, $k = 1$. Find LB. Go to Step 1a.

Step 1a: (Node qualification) If $f(k,\beta) = 0$ then go to Step 1c. Else, if $f(k,\beta) \le f(k,\beta')$ for any $\beta' < \beta$, then go to Step 1c. Else, if $f(k,\beta) + LP_{(k+1,\beta)} \le LB$, go to Step 1c. Else, go to Step 1b.

Step 1b: (Reach from (k,β)) For $j = 1, 2, \ldots, n_{k+1}$, if $\beta + a_{kj} \le B$ and $f(k,\beta) + c_{kj} > f(k+1, \beta + a_{kj})$, then let $f(k+1, \beta + a_{kj}) = f(k,\beta) + c_{kj}$ and $soln(k+1, \beta + a_{kj}) = j$. Go to Step 1c.

Step 1c: Let $\beta = \beta + 1$. If $\beta < B$ then go to Step 1a. Else, go to Step 2.

Step 2: Let $k = k + 1$ and $\beta = 1$. If $k < M$, then go to Step 1a. Else, go to Step 3.

Step 3: (Find optimal solution) Let optimal value equal $\max_{1\le\beta\le B} f(M,\beta)$. Let b be the argument of this maximum. Let $k = M$ and $\beta = b$. Go to Step 4.

Step 4: (Recover solution) Let $x_{k,soln(k,\beta)} = 1, \beta = \beta - a_{k,soln(k,\beta)}, k = k - 1$. If $k > 0$ then go to Step 4. Else, stop.

3. Variable Elimination

Variable elimination methods have been used successfully to reduce the number of variables in a multiple choice integer program prior to the implementation of an algorithm. Sweeney and Murphy [1981] show that we can determine, *a priori*, that some variables cannot be in an optimal solution, and hence can be eliminated from the problem. Variable elimination can be easily implemented in the hybrid algorithm since we can quickly obtain an optimal solution to $LMK_{(k,\beta)}$.

Let $\bar{c}_{ij}$ be the optimal reduced cost for the variable x_{ij} in the linear program $LMK_{(M,B)}$. Since the multiple choice knapsack problem is a special case of the multiple choice integer

program, we can use the following results from Sweeney and Murphy directly.

Theorem 4: If $\bar{c}_{ij} < f(M,B) - LP_{(1,B)}$ then no solution with $x_{ij} = 1$ can be optimal to $MK_{(M,B)}$.

Corollary: Let LB be any lower bound on $f(M,B)$. Then if $\bar{c}_{ij} < LB - LP_{(1,B)}$ then no x with $x_{ij} = 1$ can be optimal.

This result can be implemented to accelerate the hybrid algorithm. At each stage of the algorithm, only items, j, with $\bar{c}_{ij} \geq LB - LP_{(1,B)}$ need be considered. Theorem 4 can be implemented in Step 1a of the hybrid algorithm. Note that the duality gap, $\max_{0 \leq \beta \leq B}\{f(k,\beta) + LP(k+1, B-\beta)\} - LP_{(1,B)}$, gets smaller as k increases, therefore, elimination should not only occur only at the beginning of the algorithm, but immediately prior to reaching from each multiple choice set.

Implementation of variable elimination requires little information beyond that calculated for the hybrid algorithm. The only additional work is solving $LMK_{(M,B)}$ to obtain $\bar{c}_{ij}$. The algorithm for $LP_{(k+1,B-\beta)}$ works for this as well.

Table 2 shows the percentage of variables eliminated for the set of randomly generated test problems described in Section 4. On average, 76% of the variables were eliminated due to this result.

4. Computational Tests

4.1 Randomly Generated Problems

The algorithms above were coded in C. Computational tests were run as described in Sinha and Zoltners. Times reported in Table 1 are in milliseconds on a VAX 780. Times reported are the median computation times for randomly generated problems for each of three algorithms: Sinha and Zoltners implementation of branch and bound (25 problems per cell), pure dynamic programming (Algorithm 1, 40 problems per cell) and the hybrid algorithm (Algorithm 2) with variable elimination (40 problems per cell). Although Sinha and Zoltners times were obtained on a different computer, their times were converted to equivalent times using a conversion factor of obtained from Dongarra [1987]. The values

are an estimate of what computation times might be were their algorithm run on a VAX 780.

TABLE 1

The characteristics of each cell are M, n_i (constant for all i), and $max\ a$, where a_{ij} and c_{ij} were uniformly distributed over the integers $1, 2, \ldots, max\ a$. As in Sinha and Zoltners, $max\ a$ was set at $2n_i$ and $8n_i$ for each combination of M and n_i.

The hybrid algorithm shows substantial computational savings over dynamic programming. The times obtained are of the same order of magnitude as Sinha and Zoltners branch and bound times.

Table 2 shows the percentage of variables eliminated during solution of the test problems. The columns labeled "constant" correspond to a single elimination pass at the beginning of the algorithm. Those labeled "variable" allow repeated elimination as the duality gap is narrowed.

TABLE 2

4.2 The Homart Problems

In a project with Homart Development Co. we sought optimal merchandising plans for shopping malls. One subproblem involved analysis of a single subspace of a mall, for example, the area near the central fountain. Given a list of potential stores, e.g., jewelry store, women's apparel store, etc., choose the set of stores that maximizes the return to the developer for this area. Each store could be set at several discrete sizes and the size must be chosen concurrently with store selection. The discretization of sizes is caused by the nonlinear relationship of return to size.

In each problem, each multiple choice set corresponds to a potential store. Each item in that multiple choice set is a potential size for that store. If multiple stores of one type are possible in this space, for example jewelry stores, a multiple choice set is constructed for each possible store. The maximum number of stores of any type is set by market and policy constraints. Then a_{ij} is the number of square feet taken by a store, i, if set at size j. And c_{ij} is the return to the developer of scenario ij. The overall size of the area to be planned is B.

The appropriate formulation here is not exactly the multiple choice knapsack as defined. It is possible that some potential stores are not placed in this area at any size. Hence, the multiple choice constraints should be inequalities. This can be handled by adding a dummy variable to each multiple choice set. It is computationally more efficient to alter the algorithm to allow inequality interpretation.

A generalization of $MK_{(M,B)}$ in which at most one item is chosen from each multiple choice set requires modification of equation (2) to

$$f(k,\beta) = \max\{f(k-1,\beta), \max[f(k-1,\beta-a_{kj}) + c_{kj} : a_{kj} \leq \beta]\}.$$

This equation results in inclusion of an item from the k-th multiple choice set only if it results in a higher value knapsack than one of the same weight with fewer items included.

The problems tested represent six areas from three actual malls owned by Homart. These problems are very difficult for dynamic programming based codes due to their large right hand side values; as large as 28,509. In all cases, three sizes were considered for each store, i.e., $n_i = 3$ for all i. The times reported are milliseconds on a VAX 780. Table 3 gives details for the six test problems. Results show that even for extremely difficult examples, the hybrid algorithm is capable of solving realistically sized problems.

TABLE 3

As another test, we reran the problems with all values of a_{ij} and B divided by 10. Since the $\{a_{ij}\}$ and B must be integers, we truncated B and rounded the $\{a_{ij}\}$. The objective of such scaling is reduction of computation time. The potential danger is loss of optimality and/or feasibility. On average, computation times were reduced by 11%. An optimal solution to the original problem was discovered in four of the six problems. The other two resulted in less than one half percent error. All solutions found were feasible in the original problems.

5. Conclusions and Extensions

This paper presents a simple dynamic programming formulation for the multiple choice knapsack problem by defining and solving multiple choice knapsack functions. Computational tests are competitive with the best published techniques. An advantage of this

technique, relative to the common branch-and-bound algorithms, is its ease of coding. The code is available from the authors.

REFERENCES

Bean, J. [1984], "A Lagrangian Algorithm for the Multiple Choice Integer Program," *Operations Research*, Vol. 32, pp. 1185–1193.

Bean, J., C. Noon, S. Ryan and G. Salton [1987], "Selecting Tenants in a Shopping Mall," *Interfaces*, Vol. 18, pp. 1–9.

Bean, J. and R. Smith [1985], "Optimal Capacity Expansion Over an Infinite Horizon," *Management Science* , Vol. 31, pp. 1523–1532.

Dantzig, G. and R. Van Slyke [1967], "Generalized Upper Bounding Techniques," *Journal of Computer and System Science*, Vol. 1, pp. 213-226.

Denardo, E. [1982], *Dynamic Programming: Models and Applications*, Prentice-Hall, Englewood Cliffs, NJ.

Dongarra, J. [1987], "Performance of Various Computers Using Standard Linear Equations Software in a Fortran Environment," Argonne National Laboratory, Argonne, Illinois.

Dudziński, K. [1984], "A Dynamic Programming Approach to Solving the Multiple Choice Knapsack Problem," *Bulletin of the Polish Academy of Sciences*, Vol. 3, pp. 205–209.

Dyer, M., N. Kayal, and J. Walker [1984], "A Branch and Bound Algorithm for Solving the Multiple-Choice Knapsack Problem," *Journal of Computational and Applied Mathematics*, Vol. 11, pp. 231-249.

Gilmore, P. and R. Gomory [1966], "The Theory and Computation of Knapsack Functions," *Operations Research*, Vol. 14, pp. 1045–1074.

Lesso, W. [1970], "Optimal Selection of Public and Private Projects in Water Resource Systems," Operations Research Group, The University of Texas at Austin.

Morin, T., and R. Marsten [1976], "Branch-and-Bound Strategies for Dynamic Programming," *Operations Research*, Vol. 24, pp. 611–627.

Nauss, R. [1978], "The $0-1$ Knapsack Problem with Multiple Choice Constraints," *European Journal of Operational Research*, Vol. 2, pp. 125–131.

Sinha, P. and A. Zoltners [1979], "The Multiple Choice Knapsack Problem," *Operations*

Research, Vol. 28, pp. 503–515.

Sweeney, D., and R. Murphy [1981], "Branch and Bound Methods for Multi-item Scheduling," *Operations Research*, Vol. 29, pp. 853–864.

Zemel, E. [1980], "The Linear Multiple Choice Knapsack Problem," *Operations Research*, Vol. 28, pp. 1412–1423.

TABLE 1: Computational Results on Randomly Generated Problems

m	n_i	Number of variables	$0 \leq a, c \leq 2n_i$			$0 \leq a, c \leq 8n_i$		
			B&B	DP	VEHyb	B&B	DP	VEHyb
10	10	100	22	135	33	54	264	53
10	20	200	49	553	68	94	1060	97
10	50	500	121	3387	207	263	7104	307
20	10	200	54	691	65	107	1650	120
20	20	400	112	2698	135	254	6645	213
50	10	500	174	4579	184	245	14323	338

Median times on a VAX 780. The first column reports Sinha and Zoltners results (median of 25 problems) converted by a factor of 6.42/1.44 as given in the Dongerra paper on "Performance of Various Computers Using Standard Linear Equations Software in a Fortran Environment." The second column gives median times for a Algorithm 1 and the third column gives median times for Algorithm 2 with variable eliminiation (median of 40 problems).

TABLE 2: Percent of Variables Eliminated by Variable Reduction

m	n_i	Number of Variables	$0 \leq a, c \leq 2n_i$		$0 \leq a, c \leq 8n_i$	
			%Var elim constant	%Var elim variable	%Var elim constant	%Var elim variable
10	10	100	32.5	70.2	26.8	64.4
10	20	200	28.8	75.2	26.7	71.1
10	50	500	15.3	77.9	17.0	73.2
20	10	200	40.3	76.3	20.6	74.1
20	20	400	25.0	82.1	33.6	82.6
50	10	500	32.0	79.0	33.6	82.0

When variable elimination using the inital duality gap is used, the percentage of variables eliminated is given in the first column of each pair. When a variable (decreasing) duality gap is used, the percentage of variables eliminated increased significantly (second column of each pair).

TABLE 3: Computational Results for the Homart Problems

m	n_i	B	Computation Times	
			Without Var elim	With Var elim
7	3	5841	70.0	55.0
11	3	8382	385.1	383.5
12	3	17772	573.3	658.6
14	3	20592	547.8	543.1
16	3	21766	141.4	407.7
19	3	28509	491.9	334.3