MULTIPLE CHOICE KNAPSACK FUNCTIONS

James C. Bean
Department of Industrial Operations and Engineering

The University of Michigan
Ann Arbor, Michigan 48109

Technical Report 87-26

October 1987

MULTIPLE CHOICE KNAPSACK FUNCTIONS

James C. Bean[†]
Department of Industrial and Operations Engineering
The University of Michigan
Ann, Arbor, MI 48109-2117

January 4, 1988

ABSTRACT

An optimal dynamic programming algorithm is presented for the multiple choice knapsack problem. The algorithm is computationally competitive with the best published algorithms and simpler to state and code.

OR/MS Keywords: 114 Dynamic programming, deterministic; 628 Programming, integer, algorithms

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

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

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

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

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

The MK_b and its variations have been studied frequently in the literature, e.g., Zemel [1980], Glover and Klingman [1980], Sinha and Zoltners [1979], Ibaraki, et al. [1978] and Nauss [1975]. The traditional solution approach has been branch-and-bound supported by linear programming relaxations. This technique is attractive since the linear MK_b has special structural properties allowing $O(n \ln n)$ solution.

[†] This paper is based on work supported by the National Science Foundation under Grant ECS-8700836 to The University of Michigan

The literature contains mention of several applications of the structure such as capital budgeting and menu planning (Sinha and Zoltners). 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 profit from a mall by selecting an appropriate group of stores to populate the halls between the department stores. A set of potential stores is considered, each having various possible sizes. In the formulation above, i runs through the possible stores and j enumerates the sizes for store i. The knapsack constraint forces us within the square footage limit of the mall.

This paper presents a dynamic programming approach to the multiple choice knapsack based on the Gilmore and Gomory [1966] recursion for the knapsack problem. Advantages of this approach over traditional branch-and-bound techniques include stable and predictable computational performance and simple coding. It is computationally competitive with the best branch-and-bound codes. In Section 1 we develop the algorithm. Section 2 contains computational results. The third Section is a summary and conclusions.

1. Dynamic Programming Formulation

In MK_b let $n = \sum_{i=1}^{m} n_j$ be the number of variables. Rather than viewing the problem has making n $0-1$ 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 $0 \leq k \leq m$, $0 \leq \beta \leq b$. Note that this is substantially smaller than the state space for the $0-1$ knapsack problem (Salkin [1975]). The multiple choice knapsack function, $f(k, \beta)$, is the optimal value for MK_β, considering only variables in multiple choice sets 1 through k. That is, $f(k, \beta) =$

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

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

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

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

Then $f(m,b)$ is the optimal value to MK_b. We recursively solve a collection of these functions to determine $f(m,b)$.

Theorem 1: If the optimal solution to MK_b has a subcollection of items from multiple choice sets 1 through k having aggregate size β, this subcollection must be optimal for $MK_{(k,\beta)}$.

Proof: If not, replace that subcollection with the optimal solution to $MK_{(k,\beta)}$. The remainder of the solution is unaffected since the same multiple choice sets and portion of the knapsack are still available. The sum of the contributions of the two partial solutions is greater than the original, which was assumed to be optimal, a contradiction.■

The dynamic programming network implied by Theorem 1 has an arc to (k,β) from $(k-1,\beta-a_{kj})$ for $j=1,2,\ldots,n_k$ with value c_{ij}. Additional arcs extend to (k,β) from $(k,\beta-1)$, where appropriate, to allow for slack in the knapsack constraint. They are assigned value 0. We seek the longest path from state $(0,0)$ to state (m,b).

The optimal value, $f(m,b)$, is found by solving forward with the recursive equations:

$$f(1,\beta)=\max\{f(1,\beta-1),\max[c_{1j}:a_{1j}\leq\beta]\}, \tag{1}$$

$$f(k,\beta)=\max\{f(k,\beta-1),\max[f(k-1,\beta-a_{kj})+c_{kj}:a_{kj}\leq\beta,f(k-1,\beta-a_{kj})>0]\}, \tag{2}$$

where (2) is used for $k>1$.

Remark: The condition $f(k-1,\beta-a_{kj})>0$ is necessary to force selection of one element of each multiple choice set. It implies an assumption that all $c_{ij}>0$. If not, a suitable constant can be added to all variables in a multiple choice set to attain positivity without loss of optimality.

Remark: Replacing arcs to (k,β) from $(k,\beta-1)$ by arcs to (k,β) from $(k-1,\beta)$ has the effect of relaxing the multiple choice constraints to inequalities. In this case the condition $f(k-1,\beta-a_{kj})>0$ is not necessary since we need not use an item from each multiple choice set. The resulting recursive equation is:

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

This is analogous to the dynamic programming flow in the $0-1$ knapsack problem where not every variable need be chosen. This inequality formulation is appropriate in the mall merchandising example which motivated this work.

The following algorithm solves equations (1) and (2).

Algorithm

Step 0: (Initialize) Set $f(k,0) \leftarrow 0, f(0,\beta) \leftarrow 0, soln(k,\beta) \leftarrow 0, x(k,\beta) \leftarrow 0, k \leftarrow 1$. Solve (1) for $1 \leq \beta \leq b$. For each β, if $\max[c_{1j} : a_{1j} \leq \beta] > 0$, set $soln(1,\beta) \leftarrow \arg\max[c_{1j} : a_{1j} \leq \beta]$. Go to Step 1.

Step 1: (Solve equation) Let $k \leftarrow k+1$. Solve (2) for $1 \leq \beta \leq b$. For each β, if $\max[f(k-1,\beta-a_{kj})+c_{kj} : a_{kj} \leq \beta, f(k-1,\beta-a_{kj}) > 0] > f(k,\beta-1)$, set $soln(k,\beta) \leftarrow \arg\max[f(k-1,\beta-a_{kj})+c_{kj} : a_{kj} \leq \beta, f(k-1,\beta-a_{kj}) > 0]$. If $k = m$, let $\beta \leftarrow b$ and go to Step 2. Else, go to Step 1.

Step 2: (Recover solution) If $soln(k,\beta) > 0$, let $x_{k,soln(k,\beta)} = 1, \beta \leftarrow \beta - a_{k,soln(k,\beta)}, k \leftarrow k-1$. Else, let $\beta \leftarrow \beta - 1$. Go to Step 2. Stop when $k = 0$.

Consider the following small example:

$$\max 3x_{11} + 5x_{12} + 7x_{21} + 10x_{22}$$

$$\text{subject to}: 1x_{11} + 2x_{12} + 1x_{21} + 3x_{22} \leq 4$$

$$x_{11} + x_{12} = 1$$

$$x_{21} + x_{22} = 1$$

$$x_{ij} \in \{0,1\}$$

Figure 1 shows the underlying network for this example. Table 1 displays the flow of the algorithm for the example.

FIGURE 1: Dynamic Programming Network for the Example

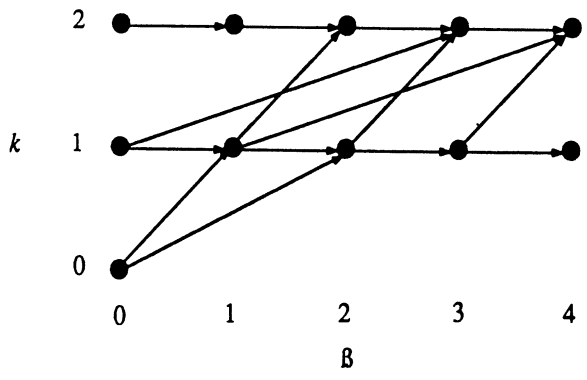

TABLE 1: Example Algorithm Flow

k	β	$f(k,\beta)$	$soln(k,\beta)$
0	0	0	0*
1	1	3	1*
1	2	5	2
1	3	5	0
1	4	5	0
2	1	0†	1
2	2	10	1
2	3	12	1
2	4	13	2*

†This node has value 0 since there is no combination of choices, one from each multiple choice set, that has total weight 1.

*Nodes along the optimal path. Indicates that the optimal solution is $x_{22} = x_{11} = 1$, others zero. Optimal value is 13.

An important result in the Gilmore and Gomory paper is proof of the existence of a knapsack size, b^*, such that for any $b \geq b^*$, there is an optimal solution containing at least one of the highest value density item, where value density is c_i/a_i. This has the computational impact of limiting the largest dynamic program that must be solved for a given collection of items. For any knapsack larger than b^*, reduce b by the size of the highest value density item and consider the smaller problem. Iterative implementation of this theorem eventually brings the right hand side down to b^* at most.

An analogous result is trivial for MK_b. Let $a_i^* = \max_j(a_{ij})$.

Lemma 2: If $b \geq \sum_{i=1}^{m} a_i^*$ then an optimal solution is $x_{ij^*} = 1$, for all i, where $j^* = \arg\max_j(c_{ij})$. All other variables equal zero.

Proof: For a b this large the knapsack constraint is redundant. The solution stated in the Lemma solves the problem with knapsack constraint relaxed.■

This result is important when viewing the computational bounds in Section 2.

2. Computational Tests

The algorithm above was coded in fortran H. Computational tests were run as described in Sinha and Zoltners. Simple variable elimination tests from Sinha and Zoltners were employed prior to executing the dynamic programming algorithm. Times reported are in milliseconds on an IBM 3090-400 at The University of Michigan, running the MTS operating system.

TABLE 2: CPU Times in Milliseconds on an IBM 3090-400

10 Randomly Generated Problems per Line

m: # sets	n_k: items/set	*maxa*: data range	Minimum CPU	Maximum CPU	Average CPU	Fitted Average CPU
10	10	20	4	5	4.6	4.5
10	10	80	18	22	19.3	19.7
10	20	40	14	17	15.4	15.6
10	20	160	62	71	67.7	67.8
10	50	100	75	84	80.2	79.6
10	50	400	339	372	357.1	346.6
20	10	20	17	20	18.9	18.8
20	10	80	78	90	84.9	81.9
20	20	40	59	67	62.0	64.6
20	20	160	261	284	271.9	281.5
50	10	20	121	131	126.3	123.5
50	10	80	516	552	533.6	537.8

Table 2 shows the results for 120 randomly generated test problems. Each cell gives the minimum, maximum and average CPU time for 10 random problems of that size. The characteristics of each cell are m, n_i (constant for all i), and *maxa*, where a_{ij} and c_{ij} were uniformly distributed over the integers $1, 2, \ldots,$ *maxa*. As in Sinha and Zoltners, *maxa* was set at $2n_i$ and $8n_i$ for each combination of m and n_i.

Lemma 3: Computation required for this algorithm is $O(nb)$.

Proof: The number of states is $O(mb)$. To label each state the computation is $O(n_k)$. Hence, overall computation for the examples is on the order of $\sum_{\beta=1}^{b} \sum_{k=1}^{m} n_k = nb$.■

A regression was run to measure stability and predictability of computational performance, fitting the model

$$CPU = K m^{\beta_m} n_k^{\beta_n} maxa^{\beta_a},$$

where K is a constant. The adjusted r^2 was 1.00 with exponents for m, n_i and *maxa* of 2.05, .72, and 1.06, respectively. Fitted CPU values are included in Table 1. Since b is approximately $m \times$ *maxa* and $n = mn_k$, these results show that, for these problems, the computational performance is highly predictable and proportional to the theoretical bounds.

Remark: The only deviation between this interpretation and the regression results is that β_n is significantly less than one. This is accounted for by the preprocessing which eliminated some variables and reduced the effective n_k.

The computation times in Table 2 are of the same order of magnitude as those reported in Sinha and Zoltners, with each algorithm "winning" a subset of the cells. Variations may not be significant considering the differences in machines and operating systems. The dynamic programming approach does show an increase in computation time with b that is not so pronounced for branch-and-bound codes. This can be dealt with in two ways. As shown in the previous Section, there is an effective maximum on b due to Lemma 2. Further, computation can be saved by scaling the a_{ij} and b values. There is some loss in precision since they must take integral values. However, in many real applications the data is not known to a high precision and/or the knapsack constraint has some flexibility. For example, in the mall merchandising problem, such scaling can result in a one to two percent over or under subscription of the mall when the optimal solution is evaluated in the true knapsack constraint. All store square footages are then rescaled to find a satisfactory solution.

3. Summary and Conclusions

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 and computation times more predictable. Another advantage of this technique relative to the common branch-and-bound algorithms is its ease of coding and teaching.

Acknowledgment: I would like to thank Gary J. Salton of Homart Development Co. for motivating and supporting this work.

REFERENCES

Bean, J. [1984], "A Lagrangian Algorithm for the Multiple Choice Integer Program," **Operations Research**, Vol. 32, No. 5.

Bean, J., C. Noon, S. Ryan and G. Salton [1987], "Selecting Tenants in a Shopping Mall," Technical Report 87–1, Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, Michigan 48109. To appear in **Interfaces**.

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

Glover, F. and D. Klingman [1979], "An $O(n \ln n)$ Algorithm for LP Knapsacks with GUB Constraints," **Mathematical Programming**, Vol. 17, pp. 345–361.

Ibaraki, T., T. Hasegawa, K. Teranaka, J. Iwase [1978], "The Multiple Choice Knapsack Problem," **Journal of the Operations Research Society of Japan**, Vol. 21, No. 1, pp. 59–93

Nauss, R. M. [1975], "The $0-1$ Knapsack Problem with Multiple Choice Constraints," Working Paper, School of Business Administration, University of St. Louis.

Salkin, H. [1975], **Integer Programming**, Addison-Wesley, Reading, MA.

Sinha, P. and A. Zoltners [1979], "The Multiple Choice Knapsack Problem," **Operations Research**, Vol. 28, pp. 503–515.

Zemel, E. [1980], "The Linear Multiple Choice Knapsack Problem," **Operations Research**, Vol. 28, No. 6, pp. 1412–1423.