

A Dynamic Programming Approach to the Multiple-Choice Multi-Period Knapsack Problem and the Recursive APL2 Code

Edward Yu-Hsien Lin line@ntut.edu.tw

Department of Business Management
National Taipei University of Technology
Taipei, Taiwan

*Presented at the 17th Triennial Conference
of the International Federation of Operational Research Societies*

Abstract

The multiple-choice multi-period knapsack problem sits in the interface of multiple choice programming and knapsack problems. Previous studies of this problem had attempted to find its optimal solution through the branch-and-bound procedure using special-ordered-sets. In this paper, we propose another solution approach based on the dynamic programming to locate its optimal solution through the evaluation of Bellman's equation at each period. We also introduce a set of concise computer code written in IBM's APL2 that solves the problem recursively. Based on this developed APL2 codes, a computational experiment was conducted to further examine the performance of this dynamic programming solution approach.

Keywords: Knapsack problem; Dynamic programming; Multiple choice programming; APL2 Computing Language

*This research is supported by the National Science Council of Taiwan,
Grant Number: NSC93-2213-E-027-018*

Introduction

Both standard knapsack problem (i.e., maximize $\sum_{j \in N} c_j x_j$ subject to $\sum_{j \in N} c_j x_j \leq b$ where $N = \{1, 2, \dots, n\}$) and various types of non-standard knapsack problems have been widely studied by researchers leading to numerous publications.^{1,2} In our bibliographical survey on many well-known non-standard knapsack problems,³ a non-standard knapsack problem called the *multiple-choice multi-period knapsack problem* (MMKP) was introduced which has the form of:

$$(P) \quad \text{Maximize} \quad \sum_{i=1}^m \sum_{j=1}^{n_i} c_{ij} x_{ij}$$

$$\text{subject to:} \quad \sum_{t=1}^i \sum_{j=1}^{n_t} w_{tj} x_{tj} \leq b_i \quad i \in M = \{1, 2, \dots, m\}, \quad (1)$$

$$\sum_{j=1}^{n_i} x_{ij} = 1 \quad i \in M, \quad (2)$$

$$x_{ij} \in \{0, 1\}, \quad i \in M, \quad j \in \{1, 2, \dots, n_i\}, \quad (3)$$

$$n_i \in \{1, 2, \dots\},$$

$$b_1 < b_2 < \dots < b_m.$$

An MMKP formulated in the form of (P) has $\sum_{i \in M} n_i$ variables scattered in m periods with its knapsack capacity expands over these m periods to accommodate the n_i additional candidate items entered into period i , $i \in M$. The MMKP seeks to select one item from each period to maximize the overall return covering these m periods. Its application includes the modeling of multi-period capital budgeting with multiple choice requirement,^{4,5} the multi-job scheduling for single machine with deadlines,⁶ etc.

The MMKP can be regarded as an extension of the *multi-period knapsack problem* (i.e., problem (P) without the existence of constraints (2) and (3)) which has been studied by Faaland⁷ and by Dudzinski and Walukiewicz.⁸ One can also view MMKP as a variant of the *multiple choice nested knapsack problem* studied by Armstrong et al.⁹ with period i being the subset of period $i+1$, $i = 1, 2, \dots, m-1$.

A solution approach to MMKP was recently proposed by Lin and Wu.¹⁰ Based on the primal and dual gradient methods, Lin and Wu first introduced a heuristic approach to obtain a strong lower bound for (P). Using this lower bound as an initial solution, they then developed two branch-and-bound procedures based on the branching scheme for special-ordered-sets type 1 variables in non-convex programming (see Beale and Tomlin¹¹ for more details) to locate the optimal solution. The first branch-and-bound

procedure solves the candidate problem using a special algorithm derived from the concept for nested knapsack problems.⁹ The second branch-and-bound procedure solves the candidate problem using the generalized upper bounding technique¹² through the concept of multiple choice programming.¹³

In this paper, we propose another solution approach based on dynamic programming approach to directly locate the optimal solution of an MMKP through recursive calculation. Dynamic programming has been incorporated in the solving of some non-standard knapsack problems such as the multiple-choice knapsack problem¹⁴ and the multidimensional knapsack problem.¹⁵ While it is understandable that dynamic programming approach is usually impractical for solving the large-scale problems due to the necessity of accommodating a large number of possible state variables (i.e., identification, storage and calculation), it does provide some benefits such as the quick and clear identification of all alternative optimal solutions.

We have also developed a computer program written in IBM's APL2¹⁶ to test and to verify our proposed solution approach. This computer program is available to the interested readers upon making the request to the author.

Properties of MMKP

Based on the studies by Lin and Wu,¹⁰ MMKP formulated as (P) possess the following properties:

Simple Dominance: If $c_{ir} \geq c_{is}$ and $w_{ir} \leq w_{is}$, $i \in M$ and $r, s \in \{1, 2, \dots, n_i\}$, then $x_{is} = 0$ in the optimal solution to (P).

Convex Dominance: For an MMKP formulated as (P), if

$$(c_{is} - c_{it}) / (w_{is} - w_{it}) \leq (c_{ir} - c_{is}) / (w_{ir} - w_{is})$$

where $i \in M$, $r, s, t \in \{1, 2, \dots, n_i\}$, and $t < s < r$, then $x_{is} = 0$ in the optimal solution to the LP relaxation of (P) (i.e., (P) with $x_{ij} \in \{0, 1\}$ replaced by $0 \leq x_{ij} \leq 1$, $i \in M$, $j \in \{1, 2, \dots, n_i\}$).

The size of a given MMKP can be reduced and its variables reorganized to have $c_{ij} < c_{i,j+1}$ and $w_{ij} < w_{i,j+1}$, $i \in M$, $j \in \{1, 2, \dots, n_i-1\}$ based on *Simple Dominance*. Furthermore, *Convex Dominance* allows one to remove convex-dominant variables from (P) and rearrange the remaining variables in an descending order of the gradients within each multiple choice set i , $i \in M$, in (P) as

$$(c_{ij} - c_{i,j-1}) / (w_{ij} - w_{i,j-1}) > (c_{i,j+1} - c_{ij}) / (w_{i,j+1} - w_{ij}), \quad j = 2, 3, \dots, n_i. \quad (4)$$

LP Property: If the variables in (P) are neither simple-dominant nor convex-dominant and satisfy condition (4), then for every multiple choice set of (P), at most two variables can have fractional values in the optimal LP solution to (P). Furthermore, these two variables must be adjacent.

Lin and Wu define an indicator called the *coefficients of tightness* associated with period i , $i \in M$ as

$$\rho_i = ((\sum_{t=1}^i w_{t,n_t}) - b_i) / \sum_{t=1}^i (w_{t,n_t} - w_{t1}), \quad i \in M.$$

When the variables of a given MMKP formulated as (P) are reorganized to have $c_{ij} < c_{i,j+1}$ and $w_{ij} < w_{i,j+1}$, $i \in M$, $j \in \{1, 2, \dots, n_i-1\}$, its optimal solution $x^* = [x_1^* \ x_2^* \ \dots \ x_m^*]$ possesses the following property:

- Solution Property:** (i) If $\max\{\rho_i \mid i \in M\} \leq 0$, then $x^* = [x_{i,n_i} \mid i \in M]$.
(ii) If $\max\{\rho_i \mid i \in M\} = \rho_h = 1$, then x^* contains $[x_{i,n_i} \mid i \in \{1, 2, \dots, h\}]$.
In the case of more than one values of h , choose the largest one.
(iii) If $\max\{\rho_i \mid i \in M\} > 1$, then (P) has no feasible solution.

Since both (i) and (iii) (i.e., obvious solution and infeasible solution) in an MMKP can be easily detected by observing the maximum coefficients of tightness. One has to consider only the MMKPs with $0 < \max\{\rho_i \mid i \in M\} \leq 1$.

Without loss of generality, we will assume that there are no simple dominated variables in (P) and the variables in (P) have been arranged in the ascending order of c_{ij} and w_{ij} , $i \in M$, $j \in \{1, 2, \dots, n_i\}$.

A Review on the Previously Proposed Solution to MMKP

Assume that problem (P) has no simple-dominant variables and that variables in (P) have been reordered so that $c_{ij} < c_{i,j+1}$ and $w_{ij} < w_{i,j+1}$, $i \in M$, $j \in \{1, 2, \dots, n_i-1\}$. Furthermore, the coefficients of tightness of (P) satisfy $0 < \max\{\rho_i \mid i \in M\} \leq 1$. Based on these assumptions, Lin and Wu first introduce two heuristics to find a strong lower bound for an MMKP formulated as (P). The first heuristic, called dual gradient heuristic, starts with an infeasible solution $[x_{i,n_i} \mid i \in M]$ with the maximum objective value. It then gradually removes the degree of infeasibility by pushing this solution towards the realm of feasibility following the slowest gradient path to ensure the minimum possible degradation of the objective value. A lower bound to (P) can be obtained when the

feasibility is reached. The second heuristic, called primal gradient heuristic, starts with a clear feasible solution $[x_{1n_i} | i \in M]$ with the minimum objective value. It then repeatedly improves the solution quality by pushing this solution towards the rim of the feasible region following the steepest gradient path to achieve the maximum possible enhancement of the objective value. A lower bound to (P) can be obtained immediately precedent to the encounter of the infeasibility. The concept of these two heuristics is analogous to the ones proposed by Senju and Toyoda¹⁷ and by Toyoda¹⁸ for the multi-dimensional knapsack problem, respectively.

Since variables in the optimal solution to (P) tend to lie in the front portion of most of the periods when $\rho_i, i \in M$ have high values, Lin and Wu suggest that primal gradient heuristic should be adopted for tightly structured (P). On the other hand, variables in the optimal solution tend to lie in the rear portion of most of the periods when $\rho_i, i \in M$ have lower values, dual gradient heuristic is recommended when (P) is loosely structured.

Lin and Wu then develop two branch-and-bound procedures to locate the optimum of an MMKP. Both solution procedures adopt the partitioning strategy for the non-convex problems with special ordered sets of type 1 variables.^{19,20} The first branch-and-bound procedure solves the candidate problem using a special algorithm based on the *LP Property* of (P) to find its LP solution. The partitioning point is selected as the one that separates the two variables having the fractional value solutions. The second branch-and-bound procedure solves the candidate problem through the GUB technique.¹² The partitioning point in this case is chosen using the weighted mean method as suggested by Beale and Tomlin¹¹. To expedite the convergence, penalty calculation^{21,22,23} is conducted to strengthen the bound. The second branch-and-bound procedure is clearly based on the multiple choice programming.

The computational experiment conducted by Lin and Wu on an IBM ES9000/320 computer using IBM's APL2 language shows that the lower bound obtained through the heuristic approach can serve as a near-optimal solution when the given MMKP is tightly structured. They also recommend that the branch-and-bound procedure using the special algorithm is used to solve the large size MMKP. On the other hand, when the given MMKP is not large, solve it through the multiple choice programming based branch-and-bound procedure.

The Dynamic Programming (DP) Based Solution Approach

Dynamic Programming (DP) decomposes a decision-making problem into several interrelated subproblems. The decision maker can then observe the dependency of the optimal values generated by one subproblem based on the possible inputs received by this subproblem. These optimal values are then incorporated in the next subproblem to determine the optimal values for both subproblems combined. Continuing this evaluation recursively to finally include all subproblems, the optimal value of the overall problem can be obtained. Since these subproblems are interrelated, the optimal solution can be traced through the interrelationship of these subproblems known as “backtracking”. Ever since the introduction of DP by Bellman,²⁴ subsequent and extensive studies using the DP technique were developed thereafter and the applications of DP can be found among many types of problems in science, engineering, and business.²⁵ The DP technique has also been directly adopted to solve the non-standard knapsack problems such as the multiple-choice knapsack problem¹⁴ and the multidimensional knapsack problem.¹⁵

When solving an MMKP using DP technique, we first construct an “m-Stage” DP structure as shown in Figure 1 with period i (i.e., multiple choice set i) corresponds to Stage $m-i+1$, $i \in M$. The decision variables, d_i , at Stage i are defined as $d_i = \{w_{i1}, w_{i2}, \dots, w_{in_i}\}$, $i \in M$. The state variables, S_i , $i \in M$, that connect these m periods (stages) are defined through a transformation function $T_i(S_i, d_i)$ as follows:

$$S_m = b_m$$

$$S_{i-1} = S_i - d_i \text{ and greater than } b_m - b_{m-(i-1)}, \quad i = m, m-1, \dots, 2, 1.$$

On the other hand, the return function, $R_i(S_i, d_i)$, $i \in M$, is defined as $R_i(S_i, d_i) = \{c_{ij} \mid j = 1, 2, \dots, n_i\}$. In this case the Bellman’s equation and its local optima over the decision variables at each stage can be derived through

$$f_i(S_i, d_i) = R_i(S_i, d_i) + f_{i-1}^*(S_i)$$

where $f_i^*(S_i) = \text{Maximum } f_i(S_i, d_i) \text{ over } d_i, \quad i \in M.$

The restrictions set on the eligibility of state variables, S_{i-1} , $i = m, m-1, \dots, 2, 1$, in the definition of transformation function for S_{i-1} is a key factor towards the optimal solution of an MMKP. Such a restriction (i.e., $S_{i-1} \geq b_m - b_{m-(i-1)}$, $i = m, m-1, \dots, 2, 1$) is necessary to ensure the satisfaction of constraints (1) in (P). That is, to fulfill the minimum requirement on the expanded knapsack capacity at each period.

< Place Figure 1 about here >

A Numerical Illustration

Consider the following MMKP with 16 variables expands over four periods.

Maximize

$$[1 \ 12 \ 18 \ 24 \ M3 \ 11 \ 18 \ 21 \ 24 \ 26 \ M6 \ 11 \ 16 \ 22 \ 25 \ M4 \ 8 \ 10] \ X$$

Subject to:

$$\begin{bmatrix} 3 & 5 & 8 & 12 & M0 & 0 & 0 & 0 & 0 & 0 & M0 & 0 & 0 & 0 & 0 & M0 & 0 & 0 \\ 3 & 5 & 8 & 12 & M8 & 9 & 11 & 14 & 18 & 21 & M0 & 0 & 0 & 0 & 0 & M0 & 0 & 0 \\ 3 & 5 & 8 & 12 & M8 & 9 & 11 & 14 & 18 & 21 & M3 & 4 & 6 & 11 & 14 & M0 & 0 & 0 \\ 3 & 5 & 8 & 12 & M8 & 9 & 11 & 14 & 18 & 21 & M3 & 4 & 6 & 11 & 14 & M2 & 4 & 6 \end{bmatrix} X \leq \begin{bmatrix} 10 \\ 18 \\ 26 \\ 32 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 & M0 & 0 & 0 & 0 & 0 & 0 & M0 & 0 & 0 & 0 & 0 & M0 & 0 & 0 \\ 0 & 0 & 0 & 0 & M1 & 1 & 1 & 1 & 1 & 1 & M0 & 0 & 0 & 0 & 0 & M0 & 0 & 0 \\ 0 & 0 & 0 & 0 & M0 & 0 & 0 & 0 & 0 & 0 & M1 & 1 & 1 & 1 & 1 & M0 & 0 & 0 \\ 0 & 0 & 0 & 0 & M0 & 0 & 0 & 0 & 0 & 0 & M0 & 0 & 0 & 0 & 0 & M1 & 1 & 1 \end{bmatrix} X = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$X = [x_{11}, x_{12}, x_{13}, x_{14}, x_{21}, x_{22}, x_{23}, x_{24}, x_{25}, x_{26}, x_{31}, x_{32}, x_{33}, x_{34}, x_{35}, x_{41}, x_{42}, x_{43}]^T = 0, 1$$

With the decision variables defined as

$$d_4 = \{3, 5, 8, 12\},$$

$$d_3 = \{8, 9, 11, 14, 18, 21\},$$

$$d_2 = \{3, 4, 6, 11, 14\}, \text{ and}$$

$$d_1 = \{2, 4, 6\},$$

the state variables can be derived as

$$S_4 = 32$$

$$\begin{aligned} S_3 &= S_4 - d_4 \text{ and greater than 22 (i.e., } 32 - 10) \\ &= \{24, 27, 29\} \end{aligned}$$

$$\begin{aligned} S_2 &= S_3 - d_3 \text{ and greater than 14 (i.e., } 32 - 18) \\ &= \{15, 16, 18, 19, 20, 21\}, \text{ and} \end{aligned}$$

$$\begin{aligned} S_1 &= S_2 - d_2 \text{ and greater than 6 (i.e., } 32 - 26) \\ &= \{6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18\} \end{aligned}$$

Conduct Bellman's forward calculation

Stage 1:

$d_1 \backslash S_1$	2	4	6	$f_1^*(S_1)$	d_1^*	$S_1 - d_1^*$
6	4	8	10	10	6	0
7	4	8	10	10	6	1
8	4	8	10	10	6	2
9	4	8	10	10	6	3
10	4	8	10	10	6	4
11	4	8	10	10	6	5
12	4	8	10	10	6	6
13	4	8	10	10	6	7
14	4	8	10	10	6	8
15	4	8	10	10	6	9
16	4	8	10	10	6	10
17	4	8	10	10	6	11
18	4	8	10	10	6	12

Stage 2: Evaluate only the entries where $S_2 - d_2 \geq 6$.

Those entries that are not applicable are indicated by "n/a".

$d_2 \backslash S_2$	3	4	6	11	14	$f_2^*(S_2)$	d_2^*	$S_2 - d_2^*$
15	16 (6+10)	21 (11+10)	26 (16+10)	n/a	n/a	26	6	9
16	16 (6+10)	21 (11+10)	26 (16+10)	n/a	n/a	26	6	10
18	16 (6+10)	21 (11+10)	26 (16+10)	32 (22+10)	n/a	32	11	7
19	16 (6+10)	21 (11+10)	26 (16+10)	32 (22+10)	n/a	32	11	8
20	16 (6+10)	21 (11+10)	26 (16+10)	32 (22+10)	35 (25+10)	35	14	6
21	16 (6+10)	21 (11+10)	26 (16+10)	32 (22+10)	35 (25+10)	35	14	7

Stage 3: Evaluate only the entries where $S_3 - d_3 \geq 14$.

$d_3 \backslash S_3$	8	9	11	14	18	21	$f_3^*(S_3)$	d_3^*	$S_3 - d_3^*$
24	29 (3+26)	37 (11+26)	n/a	n/a	n/a	n/a	37	9	15
27	35 (3+32)	43 (11+32)	44 (18+26)	n/a	n/a	n/a	44	11	16
29	38 (3+35)	46 (11+35)	50 (18+32)	47 (21+26)	n/a	n/a	50	11	18

Stage 4: Evaluate only the entries where $S_4 - d_4 \geq 22$.

$S_4 \backslash d_4$	3	5	8	12	$f_4^*(S_4)$	d_4^*	$S_4 - d_4^*$
32	51 (1+50)	56 (12+44)	55 (18+37)	n/a	56	5	27

Backtracking

Backtracking from Stage 4 to Stage 1 by observing d_i^* and $S_i - d_i^*$, $i = 4, 3, 2, 1$, the optimal solution can be obtained as $d_4^* = 5$; $d_3^* = 11$; $d_2^* = 6$; $d_1^* = 6$. That is, $x_{12} = 1$; $x_{23} = 1$; $x_{33} = 1$; $x_{43} = 1$ with the optimal objective value of 56 (i.e., $f_4^*(S_4=32)$).

Alternate Optimal Solutions

If we change the objective coefficient c_{22} from 11 to 12, the forward calculation of Stage 3 and Stage 4 would turn into the followings.

Stage 3:

$S_3 \backslash d_3$	8	9	11	14	18	21	$f_3^*(S_3)$	d_3^*	$S_3 - d_3^*$
24	29 (3+26)	38 (12+26)	n/a	n/a	n/a	n/a	38	9	15
27	35 (3+32)	44 (12+32)	44 (18+26)	n/a	n/a	n/a	44	9 or 11	18 or 16
29	38 (3+35)	47 (12+35)	50 (18+32)	47 (21+26)	n/a	n/a	50	11	18

Stage 4:

$S_4 \backslash d_4$	3	5	8	12	$f_4^*(S_4)$	d_4^*	$S_4 - d_4^*$
32	51 (1+50)	56 (12+44)	56 (18+38)	n/a	56	5 or 8	27 or 24

Under this situation, three alternate optimal solutions can be easily detected through backtracking as $d_4^* = 5$; $d_3^* = 9$; $d_2^* = 11$; $d_1^* = 6$. (i.e., $x_{12} = 1$; $x_{22} = 1$; $x_{34} = 1$; $x_{43} = 1$), or $d_4^* = 5$; $d_3^* = 11$; $d_2^* = 6$; $d_1^* = 6$. (i.e., $x_{12} = 1$; $x_{23} = 1$; $x_{33} = 1$; $x_{43} = 1$) or $d_4^* = 8$; $d_3^* = 9$; $d_2^* = 6$; $d_1^* = 6$. (i.e., $x_{13} = 1$; $x_{22} = 1$; $x_{33} = 1$; $x_{43} = 1$) all with the optimal objective value of 56.

The Development of Recursive Computer Code for MMKP

Unlike most other algorithms in optimization, DP is generally considered as an “approach” to an optimization problem, rather than an “algorithm” which provides a clear step-by-step solution procedure (a vital criterion for software programming) converging to the optimal solution of the problem. Although there have been some attempts to develop a general DP code,^{26,27} in essence, no general DP code has ever existed. This specificity of the DP technique often makes its user, particularly the novice, consider DP as a complex modeling tool in optimization. In this section, we show how we can cope with the difficulty of implementing the DP solution approach to MMKP by using the recursive capability provided by APL computing language.

APL is the acronym for “A Programming Language” invented by Dr. Kenneth Iverson at Harvard University.²⁹ It is a profoundly eloquent computing language that is directly machine executable. Its flexibility and capability of manipulating the matrices in complex ways has made APL an ideal tool for analyzing the algorithms in operational research. APL2 is an IBM product with service provided to accommodate various computing environments. For the insights of APL, please refer to the article by Sneiedovich³⁰ and the popular APL book by Gilman and Rose³¹. Interested readers can also visit the website of www-306.ibm.com/software/awdtools/apl for the details of IBM’s APL2.

A computer program is generally referred to as a “function” in APL terminology. Parameters used in an APL function are called “arguments” that are placed before and/or after the function name. The engine that drives our developed APL2 code for the optimal solution of MMKP is function **F** that performs the recursive calculation on Bellman’s equation. The pseudo code of function **F** with argument **N** representing the stage number (i.e., period) can be expressed as:

F N

- [1] Go to [4] if $N = 0$
- [2] Calculate the maximum of $R_N(S_N, d_N) + (\mathbf{F} \ N-1) T_N(S_N, d_N)$ over d_N
- [3] Stop
- [4] Define the value of **F 0**

When function **F N** is activated, it is forced to call itself as statement [2] is invoked. When this happens, the system tentatively stores the results obtained at that point and activate **F N-1**. This, in turn, activates **F N-2** and stores the tentative results followed by activating **F N-3** with the tentative results stored, etc. The process continues until it encounters **F 0**. The system then goes to statement [4] to retrieve the value of **F 0** and

incorporate it with the stored results that were tentatively halted at statement [2]. Such a recursive calculation with the calling of an executing function from within its embedded statements is a unique feature of APL. This unique capability of APL allows us to develop a set of concise and elegant APL2 code as shown in Figure 2.

< Place Figure 2 about here >

The immediate executable main function provided by this set of APL2 code is function **SOLUTION** that solves an MMKP with two arguments. The left argument is a two by two matrix with the first row representing n_i , $i \in M$ (i.e., the number of variables in each set), and the second row representing b_i , $i \in M$ (i.e., the knapsack capacity at each period). The right argument is also a two by two matrix with the first row representing c_{ij} , and the second row representing w_{ij} , $i \in M, j \in \{1, 2, \dots, n_i\}$. It should be noted that function **SOLUTION** is capable of capturing all alternate optimal solution, if exist, through the supporting function **OPTIMIZE**. The illustration of solving our two numerical examples appeared in the previous section through the execution of function **SOLUTION** is shown in Figure 3. For the pedagogical purpose, in the development of our DP code to MMKP, we have also constructed few additional APL2 functions that can solve an MMKP with the results printed out stage by stage so that users can conduct backtracking to explore more details in the solution process. All of our APL2 codes are available to the interested readers upon making request to the author.

< Place Figure 3 about here >

Computational Experiment

By using the developed APL2 codes, we conducted a computational experiment on a PC with *Pentium III* processor to examine the performance of our proposed DP approach to MMKP. Due to the general understanding that the efficiency of DP approach deteriorates quickly as the problem size increases, this computational experiment aims only at the observation of its performance on different structured MMKP.

To test problems were generated randomly with the objective coefficients c_{ij} and the constraint coefficients w_{ij} , $t \in \{1, 2, \dots, i\}$, $i \in M, j \in \{1, 2, \dots, n_i\}$, from a uniform distribution between 1 and 100. The right-hand-sides, b_i , of constraints (1) are obtained from a normal distribution with mean, μ , and standard deviation, σ , equal to

$$\mu = \theta (\max \{w_{ij}\} + \min \{w_{ij}\}) \text{ where } 0 < \theta < 1,$$

$$\sigma = 0.25 (\max \{w_{ij}\} + \min \{w_{ij}\}), t \in \{1, 2, \dots, i\}, i \in M, j \in \{1, 2, \dots, n_i\}.$$

This arrangement allows us to tighten or loosen the structure of the test problems through the control of θ . (i.e., Larger value of θ generates loosely structured test problem whereby smaller value of θ leads to tightly structured test problem). In order to assess the overall degree of tightness of an MMKP, we also use the mean of ρ_i , $i \in M$, as an *aggregate coefficient of tightness*, ρ , defined as

$$\rho = (\sum_{i \in M} \rho_i) / m.$$

Using this mechanism, we were able to generate hundreds of MMKP test problems with 48 variables without simple dominant variables. A total of 10 test problems with different degrees of aggregated coefficient of tightness were then selected randomly to solve with four periods and 12 variables in each set. Another 10 test problems were then randomly selected to solve with eight periods and six variables in each period. Finally, 10 randomly selected test problems were solved with 12 periods with four variables in each period. The same procedure was repeated for 80 variables and 120 variables, respectively, each separated into three categories representing different period and variable combinations (i.e., less periods with more variables versus more periods with less variables).

The results of this computational experiment is reported in Table 1 (48 variables), Table 2 (80 variables), and Table 3 (120 variables). Two major trends are observed through these three tables, namely:

1. For all sizes of problems, it takes longer to solve loosely structured MMKP (represented by smaller ρ values) than to solve tightly structured MMKP (represented by larger ρ values).
2. For the same problem size (i.e., same number of variables), it takes longer to solve the MMKP with more periods than those with less periods. This solution time tends to increase drastically as the problem size increases.

The first observation can be justified by that a loosely structured MMKP contains a wider solution space to search from and will, thus, created more state variables in DP approach. The second observation can also be justified because more periods represent more stages in DP approach. Consequently, it will consume more computation time in forward calculation as well as in backtracking for optimal solution.

Under the current computing environment using PC, the computation times for these test problems are relatively fast. However, the computation time is expected to increase exponentially as both the size of the problem and the number of periods increase. As a result, DP approach to MMKP should be avoided for this type of problems.

< Place Table 1 to Table 3 about here >

Concluding Remarks

This research revisits the Multiple-choice Multi-period Knapsack Problem (MMKP) introduced by Lin³. Based on the special structure exhibited by MMKP, a Dynamic Programming (DP) solution approach is proposed in this paper.

The two major shortcomings in DP approach are the difficulty of conducting the Bellman's recursive calculation (in programming phase) and its inefficiency of handling the large-scale problem known as the "curse of dimensionality". This paper presents a set of computer codes written in IBM's APL2 computing language that shows how the first shortcoming can be coped with. The codes elegantly perform the recursive calculation on Bellman's equations and are capable of capturing all alternative optimal solutions through backtracking. The use of such computing language greatly reduces the efforts in the programming phase. In terms of the curse of dimensionality, recent report by De Farias and Van Roy that uses a linear programming approach to approximate the solution of large scale problem when solved through DP³² may be able to provide some relief on the second shortcomings of DP.

Despite these, dynamic programming remains attractive due to its less complicated procedure flow and, mainly, due its capability to capture all alternative optimal solutions during the backtracking process without having to go through further detection. The findings from this paper thus put additional contributions to the rich literature on the studies in both the knapsack problems and the dynamic programming.

Acknowledgement

This research is supported by the grant from the National Science Council of Taiwan (Grant Number: NSC93-2213-E-027-018).

References

- 1 Kellerer H, Pferschy U and Pisinger D (2004). *Knapsack Problems*. Springer (ISBN: 3-540-40286-1)
- 2 Martello S and Toth P (1990). *Knapsack problems: Algorithms and computer implementation*. John Wiley & Sons: Chichester, England.
- 3 Lin EYH (1998). A bibliographical survey on some well-known non-standard knapsack problems. *INFOR (Information Science and Operational Research)* **36**: 274-317.
- 4 Pisinger D (1995). A minimal algorithm for the multiple-choice knapsack problem. *European Journal of Operational Research* **83**: 394-410.
- 5 Sinha P and Zoltners AA (1979). The multiple choice knapsack problem. *Operations Research* **27**: 503-515.
- 6 Gens GV and Levner E (1981). Fast approximation algorithm for job sequencing with deadlines. *Discrete Applied Mathematics* **3**: 313-318.
- 7 Faaland BH (1981). The multiperiod knapsack problem. *Operations Research* **29**: 612-616.
- 8 Dudzinski K and Walukiewicz S (1985). On the multiperiod binary knapsack problem. *Methods of Operations Research* **49**: 223-232.
- 9 Armstrong RD, Sinha P and Zoltners AA (1982). The multiple-choice nested knapsack model. *Management Science* **28**: 34-43.
- 10 Lin EYH and Wu CM (2004). The multiple-choice multi-period knapsack problem. *Journal of the Operational Research Society* **55**: 187-197.
- 11 Beale EML and Tomlin JA (1970). Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables. In: Lawrence J (ed) *Proceedings of the Fifth International Conference on Operational Research*, Tavistock Publications, London 447-454.
- 12 Dantzig GB and Van Slyke RM (1967). Generalized upper bounding technique. *Journal of Computer and System Sciences* **1**: 213-226.
- 13 Lin EYH (1994). Multiple choice programming: A state-of-the-art survey. *International Transactions in Operational Research* **1**: 409-421.
- 14 Pisinger D (1995). A minimal algorithm for the multiple-choice knapsack problem. *European Journal of Operational Research* **83**: 394-410.

- 15 Bertsimas D and Demir R (2002). An approximate dynamic programming approach to multidimensional knapsack problems. *Management Science* **48**: 550-565
- 16 IBM Corporation (1992). *APL2 Version 2 Programming: Language Reference (SH21-1061)*. IBM APL Development Department, San Jose, California, USA.
- 17 Senju S and Toyoda Y (1968). An approach to linear programming with 0-1 variables. *Management Science* **15**: B196-B207.
- 18 Toyoda Y (1975). A simplified algorithm for obtaining approximate solutions to zero-one programming problems. *Management Science* **21**: 1417-1427.
- 19 Lin EYH and Bricker DL (1995). Computational comparison on the partitioning strategies in multiple integer programming. *European Journal of Operational Research* **88**: 182-202.
- 20 Lin EYH and Bricker DL (1991). On the calculation of true and pseudo penalties in multiple choice integer programming. *European Journal of Operational Research* **55**: 228-236.
- 21 Tomlin JA (1970). Branch-and-bound methods for integer and non-convex programming. In: Abadie J (ed), *Integer and Nonlinear Programming*, North-Holland: Amsterdam 437-450.
- 22 Gomory RE (1960). An algorithm for the mixed integer problem. *The RAND Corporation Report RM-2579*, Santa Monica, California, USA.
- 23 Armstrong RD and Sinha P (1974). Improved penalty calculation for a mixed integer branch-and-bound algorithm. *Mathematical Programming* **6**: 212-223.
- 24 Bellman R (1957). *Dynamic programming*, Princeton University Press: Princeton, New Jersey.
- 25 Denardo E (1982). *Dynamic programming models and applications*, Prentice-Hall: Englewood Cliffs, New Jersey.
- 26 Deuermeyer BL and Curry GL (1989). On a language for discrete dynamic programming and a microcomputer implementation. *Computers and Operations Research* **16**: 1-11.
- 27 Sniedovich M and Smart JS (1982). Dynamic programming: An interactive approach. *Journal of Mathematical Analysis & Applications* **86**: 208-220.
- 28 Lin EYH and Bricker DL (1990). Implementing the recursive APL code for dynamic programming. *APL Quote Quad* **20**: 239-250.
- 29 Iverson KE (1962). *A Programming Language*, APL Press, Pleasantville, New York.

- 30 Sniedovich M (1989). The APL phenomenon: An operational research perspective. *European Journal of Operational Research* **38**: 141-155.
- 31 Gilman L and Rose AJ (1984). APL: An Interactive Approach “Includes APL2 and APL for PC’s”. *John Wiley & Sons*, New York.
- 32 De Farias DP and Van Roy B (2003). The linear programming approach to approximate dynamic programming. *Operations Research* **51**: 850-865.

Figure 1
The m-Stage DP Structure

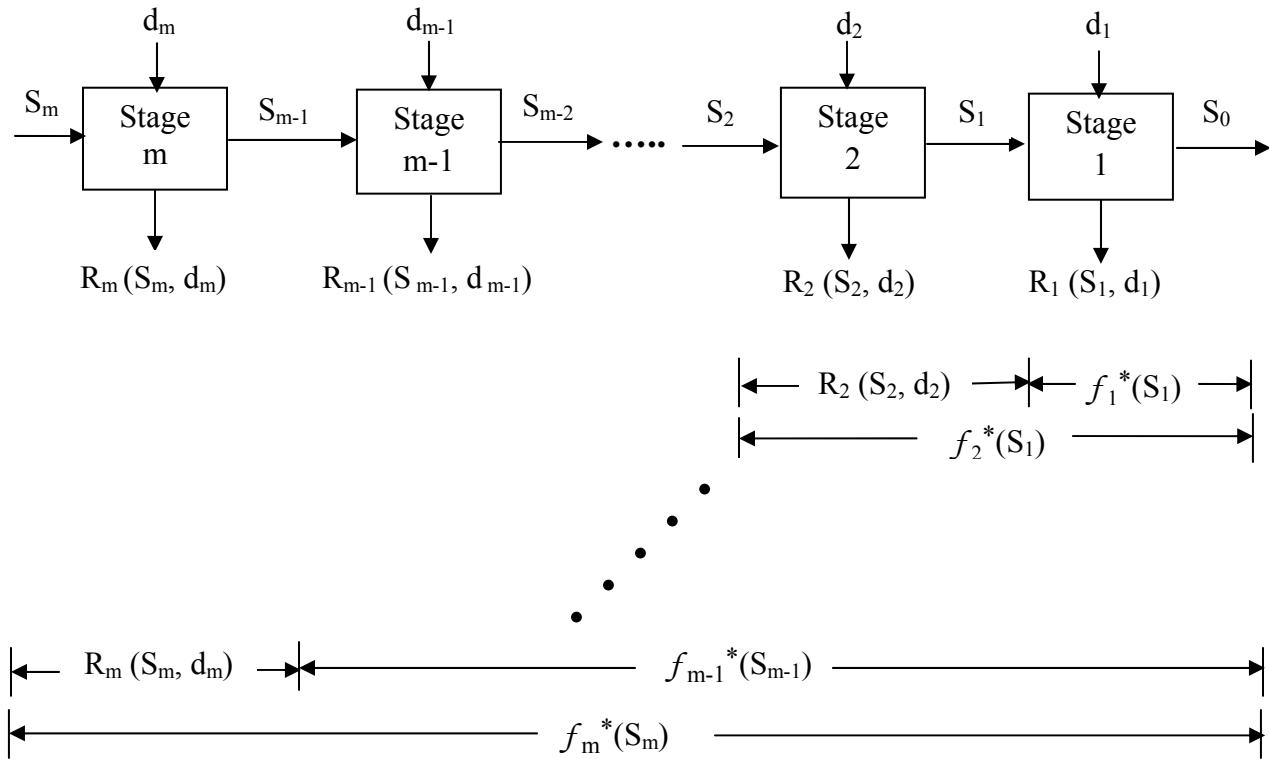


Figure 2
APL2 Codes for DP Solution to MMKP

[1] M SOLVE A;LIMIT;V;l;s;sΔ	Z+F0
[2] s+0,1sΔ ⁻¹ M[2;]	[1] Z+((ps)ρ0), ⁻ 999999
[3] 1+0	
[4] V+@BOOLEAN M[1;]	
[5] LIMIT+0,φ ⁻¹ 1+sΔ-M[2;]	DECISION L;M
[5] DECISION DP	[1] M+OPTIMIZE L
	[2] 1 1ρ' '
Z+DP;LIST	[3] ' Stage ',(9×1ρM)ρ' Decision '
[1] LIST+0 5ρ' '	[4] (4 0,(2×1ρM)ρ9 0)φ ⁻¹ 1L[;1],φM
[2] Z+Fφ ⁻¹ 1ρV	[5] 1 1ρ' '
[3] Z+((LIST[;1]) ⁻¹ LIST[;1])vLIST[;2]=sΔ)/LIST	[6] 'Optimal Value: ',φ ⁻¹ 1L[;3]
	Z+OPTIMIZE R;B;C;D;L;N;S;U
FΔ+F N;d;t	[1] Z+(0,N+[/R[;1])ρ' '
[1] →LAST IF N=0	[2] D+(B+(R[;1]=N)AR[;3]) ⁻¹ 1R[;3])/R[;4]
[2] FΔ~MAX(RETURN N)+(F N-1)(OUTPUT t+TRANSIT N)	[3] L+(0,D)⋅×((N-1)ρ0),1
[3] +0	[4] S+0,(2,ρD)ρ((ρD)ρN),B/R[;5]
[4] LAST:FΔ+F0	[5] NEXT:→0 IF 0=+/+/S
	[6] C+ ⁻¹ 1N)1L
Z+MAX F;B;I	[7] U+ ² 11S
[1] Z+[/F	[8] L+ ⁻¹ 04L
[2] I+1	[9] S+0 ⁻¹ 1S
[3] START:→NEXT IF Z[I]<1	[10] →COMPLETE IF 1=U[1]
[4] B+Z[I]⋅.=F[I;]	[11] D+(B+(R[;1]=U[1]-1)AR[;2]=U[2])/R[;4]
[5] LIST+LIST,[1]N,s[I],Z[I],φ(2,+/B)ρ(B/d),B/t[I;]	[12] L+L,[1](D⋅×(U[1]-1)=iN)+(ρD)ρ1)⋅×,C
[6] NEXT:→START IF (ps)≥I+I+1	[13] S+S,(2,ρD)ρ((ρD)ρU[1]-1),B/R[;5]
[7] END:Z+Z, ⁻ 9999999	[14] →NEXT
	[15] COMPLETE:Z+C,[1]Z
Z+RETURN N	[16] →NEXT
[1] Z+((ps)ρ1)⋅.×V[N;]/A[1;]	Z+BOOLEAN V;K;T
	[1] Z+(K⋅.≥T)-(0, ⁻¹ 1+K+V)⋅.≥T+i+/V
Z+OUTPUT T	Z+L IF C
[1] Z+s1T	[1] Z+(,C)/L
Z+TRANSIT N	
[1] Z+s⋅.-d+V[N;]/A[2;]	
[2] Z+Z+ ⁻ 9999999×-Z≥LIMIT[N]	

Figure 3
APL2 Solution to the Numerical Examples

```

      A
1 12 18 24 3 11 18 21 24 26 6 11 16 22 25 4 8 10
3  5  8 12 8  9 11 14 18 21 3  4  6 11 14 2 4  6

      M
 4  6  5  3
10 18 26 32

      M SOLVE A

Stage  Decision
  1      6
  2      6
  3     11
  4      5

Optimal Value: 56

      AΔ
1 12 18 24 3 12 18 21 24 26 6 11 16 22 25 4 8 10
3  5  8 12 8  9 11 14 18 21 3  4  6 11 14 2 4  6

      M
 4  6  5  3
10 18 26 32

      M SOLVE AΔ

Stage  Decision  Decision  Decisi
  1      6         6         6
  2     11         6         6
  3      9        11         9
  4      5         5         8

Optimal Value: 56

```

Table 1
Computational Results of Test Problems with 48 Variables

(Four periods with 12 variables in each period)

ρ	0.116	0.125	0.157	0.168	0.190	0.212	0.223	0.245	0.267	0.288
CPU	0.110	0.110	0.110	0.110	0.093	0.093	0.093	0.093	0.093	0.093
ρ	0.310	0.321	0.354	0.365	0.409	0.434	0.452	0.498	0.514	0.589
CPU	0.093	0.093	0.078	0.078	0.078	0.078	0.078	0.063	0.063	0.063
ρ	0.625	0.644	0.663	0.688	0.696	0.729	0.742	0.773	0.801	0.833
CPU	0.063	0.063	0.063	0.047	0.047	0.047	0.047	0.047	0.047	0.041

(Eight periods with six variables in each period)

ρ	0.094	0.118	0.139	0.146	0.179	0.193	0.221	0.234	0.252	0.297
CPU	0.320	0.344	0.344	0.344	0.330	0.312	0.297	0.297	0.297	0.280
ρ	0.318	0.339	0.361	0.383	0.424	0.459	0.471	0.506	0.549	0.552
CPU	0.266	0.266	0.266	0.266	0.250	0.219	0.219	0.203	0.196	0.196
ρ	0.611	0.634	0.664	0.690	0.698	0.740	0.776	0.789	0.830	0.844
CPU	0.187	0.187	0.172	0.171	0.171	0.141	0.125	0.125	0.123	0.123

(12 periods with four variables in each period)

ρ	0.094	0.125	0.147	0.161	0.176	0.193	0.225	0.244	0.279	0.299
CPU	1.359	1.250	1.186	1.108	1.078	1.031	1.022	1.000	0.986	0.921
ρ	0.319	0.354	0.370	0.387	0.429	0.446	0.479	0.513	0.548	0.582
CPU	0.860	0.829	0.781	0.745	0.703	0.681	0.628	0.563	0.516	0.516
ρ	0.615	0.641	0.651	0.678	0.699	0.720	0.743	0.750	0.786	0.832
CPU	0.437	0.437	0.437	0.437	0.422	0.398	0.319	0.297	0.281	0.257

Table 2
Computational Results of Test Problems with 80 Variables

(Five periods with 16 variables in each period)

ρ	0.092	0.145	0.154	0.177	0.198	0.211	0.222	0.251	0.270	0.281
CPU	0.219	0.219	0.219	0.210	0.187	0.187	0.187	0.141	0.140	0.140
ρ	0.305	0.322	0.358	0.362	0.396	0.411	0.461	0.464	0.517	0.587
CPU	0.141	0.138	0.125	0.125	0.110	0.110	0.110	0.110	0.094	0.078
ρ	0.613	0.624	0.646	0.677	0.708	0.730	0.759	0.783	0.809	0.836
CPU	0.063	0.063	0.063	0.062	0.062	0.062	0.062	0.047	0.047	0.047

(10 periods with eight variables in each period)

ρ	0.098	0.117	0.127	0.167	0.190	0.211	0.231	0.241	0.265	0.289
CPU	1.106	0.906	0.918	0.828	0.798	0.765	0.754	0.734	0.703	0.614
ρ	0.302	0.330	0.356	0.378	0.395	0.458	0.481	0.497	0.515	0.562
CPU	0.594	0.579	0.531	0.517	0.469	0.418	0.375	0.375	0.375	0.347
ρ	0.604	0.623	0.650	0.690	0.720	0.743	0.775	0.788	0.792	0.814
CPU	0.313	0.301	0.266	0.241	0.203	0.201	0.172	0.164	0.156	0.151

(20 periods with four variables in each period)

ρ	0.094	0.126	0.143	0.166	0.194	0.221	0.239	0.251	0.270	0.282
CPU	10.741	9.547	9.135	8.625	8.208	7.672	7.587	7.516	7.289	7.031
ρ	0.330	0.345	0.371	0.391	0.426	0.446	0.451	0.487	0.550	0.576
CPU	6.359	6.079	5.953	5.532	5.110	4.987	4.750	4.511	4.435	4.245
ρ	0.607	0.632	0.653	0.676	0.701	0.722	0.743	0.777	0.801	0.832
CPU	4.038	3.813	2.984	2.937	2.789	2.578	2.297	1.976	1.844	1.741

Table 3
Computational Results of Test Problems with 120 Variables

(Five periods with 24 variables in each period)

ρ	0.095	0.127	0.141	0.167	0.181	0.212	0.237	0.251	0.280	0.289
CPU	0.593	0.187	0.187	0.187	0.187	0.187	0.156	0.156	0.141	0.141
ρ	0.301	0.324	0.350	0.366	0.412	0.444	0.479	0.498	0.538	0.587
CPU	0.141	0.141	0.125	0.125	0.125	0.125	0.109	0.109	0.109	0.079
ρ	0.609	0.623	0.634	0.663	0.695	0.728	0.744	0.760	0.801	0.846
CPU	0.079	0.079	0.079	0.079	0.078	0.062	0.062	0.062	0.062	0.049

(10 periods with 12 variables in each period)

ρ	0.120	0.138	0.155	0.176	0.191	0.204	0.226	0.255	0.268	0.280
CPU	1.094	1.067	0.969	0.969	0.969	0.907	0.907	0.853	0.712	0.625
ρ	0.326	0.350	0.361	0.379	0.399	0.421	0.445	0.480	0.492	0.554
CPU	0.610	0.498	0.454	0.454	0.438	0.399	0.375	0.365	0.343	0.317
ρ	0.614	0.643	0.659	0.660	0.690	0.709	0.732	0.757	0.778	0.840
CPU	0.265	0.258	0.234	0.234	0.225	0.219	0.188	0.166	0.156	0.098

(20 periods with six variables in each period)

ρ	0.099	0.113	0.138	0.158	0.176	0.184	0.202	0.234	0.264	0.299
CPU	13.563	13.098	12.390	11.590	11.468	11.071	10.797	9.159	7.734	7.662
ρ	0.314	0.350	0.358	0.364	0.403	0.434	0.455	0.479	0.496	0.555
CPU	7.281	7.220	7.099	7.063	6.281	5.961	5.547	5.018	4.859	4.027
ρ	0.626	0.644	0.676	0.700	0.716	0.734	0.753	0.789	0.806	0.842
CPU	3.360	3.163	2.844	2.649	2.406	2.261	1.953	1.625	1.514	1.249