Sci. J. Fac. Sci. Menofiya Univ. Vol. VI (1992) 79-96

SOLVING INTEGER PROGRAMMING PROBLEM UTILIZING DYNAMIC PROGRAMMING

N. EL-RAMLY

Menofiya University, Shebin-El-Kom, Menofiya, Egypt.

A. EL-KASSAS

Institute of National Planning, Nasr City, Cairo, Egypt.

ABSTRACT

Many authors proposed algorithms to transform the original IP [1] problem into a Knapsack problem which may be easier to be solved by a DP[1] recurraion (*Glover* [1], *Kendel,* and *Zionts* [7]) .

Other authors transformed the IP problem into a Group optimization problem that strongly resembles the Knapsack problem . An optimal solution to the transformed problem often yield an optimal solution to the original IP problem from which it was derived (*Gomory* [2], *Shapiro,* [11]) .

Another approach is developed by mixing the methods of DP and the branch-and-bound principles . *Morin* and *Marsten* [9], has shown how branch-and-bound methods can be used to reduce storage and, possibly, computational requirements in discrete dynamic programs (*Jeromin,* and *Korner* [6], *Kovacs* [8]).

The purpose of this paper is to show the capability of utilizing Dynamic Programming (DP) in solving Integer Programming (IP) problems .

[1] IP ≡ Integer Programming, DP ≡ Dynamic Programming.

1. The Formulation of IP as a DP Model :

Consider the general integer programming problem :

(IP) : Max

$$\sum_{j=1}^{n} c_j x_j \ ,$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \ , \ i = 1, 2, \ \dots, m \ ,$$

and

$$x_j \geq 0 \ integers \ , \ j = 1, 2, \ \dots, n \ .$$

The problem (IP) can be formulated as a dynamic programming model as follows:

1. Each activity j (j = 1, 2, ..., n) may be regarded as stage (i.e. we have n-stage decision problem) .
2. The level of activity x_j (≥ 0) represents the alternatives at stage j .
3. However, m state parameters will be needed one for each constraint (i.e. the state variables are of m-dimensional) .

Let $(s_{1j} \ , \ s_{2j} \ , \ \dots \ , \ s_{mj})$ be the states of the system at stage j, that is the amounts of resources 1, 2, ..., m allocated to stage j, j+1, ..., n . Also let $f_j(s_{1j} \ , \ s_{2j} \ , \ \dots \ , \ s_{mj})$ be the optimum value of the objective function for stages j, j+1, ..., n .

The corresponding dynamic programming formulation becomes :

$$f_n(s_{1n}\ ,\ s_{2n}\ ,\ \dots\ ,\ s_{mn}) = \max_{\substack{0 \le a_{in}x_n \le s_{in} \\ ,\ i=1,\,2,\,\dots,\,m \\ 0 \le x_j \le [s_{in}/a_{in}]}} \{c_n x_n\}$$

$$f_j(s_{1j}\ ,\ s_{2j}\ ,\ \dots\ ,\ s_{mj}) = \max_{0 \le a_{ij}x_j \le s_{ij}} \{c_j x_j + f_{j+1}(s_{1j} - a_{1j}x_j\ ,\ \dots\ ,\ s_{mj} - a_{mj}x_j\ )\}$$

where i = 1, 2, ... , m ,

j = 1, 2, ... , n-1 ,

$0 \le s_{ij} \le b_i$ for all i, j ,

$0 \le x_j \le [s_{ij}/a_{ij}]$ and [y] denotes the largest integer $\le y$.

The above recursive relationship is of the backward type . Otherwise we can make the same procedure by a forward recursive equation as follow :

$$f_1(s_{11}\ ,\ s_{21}\ ,\ \dots\ ,\ s_{m1}) = \max_{0 \le x_1 \le [s_{i1}/a_{i1}]} \{c_1 x_1\}$$

$$f_N(s_{1N}\ ,\ s_{2N}\ ,\ \dots\ ,\ s_{mN}) = \max_{\substack{0 \le x_N \le [s_{iN}/a_{iN}] \\ ,\ N=1,\,2,\,\dots,\,n}} \{c_N x_N +$$

$$f_{N-1}(s_{1N} - a_{1N}x_N\ ,\ \dots\ ,\ s_{mN} - a_{mN}x_N)\}$$

where x_j , j = 1, 2, ..., n is the decision variable corresponding to stage j . This recursive scheme has a practical computational value only when the number m of state variables is small .

To demonstrate our solution procedure, consider the following simple example :

Example 1.1 :

Max $\quad Z = x_1 + x_2$

subject to $2x_1 + x_2 \leq 8$

$x_1 + x_2 \leq 8$

and $\quad x_1 \, , \, x_2 \geq 0$, integer .

Define the decision variables as $d_1 = x_1$ and $d_2 = x_2$ and let the stages correspond to the variables x_i , $i = 1,2$.

Since the problem has two constraints, there will be two state variables to search over . Denote these two state variables at the nth stage as α_n , β_n , $n = 1,2$ such that :

$$\alpha_{n-1} = \alpha_n - a_{1n} d_n \, ,$$

$$\beta_{n-1} = \beta_n - a_{2n} d_n$$

stage 1 :

$$f_1^*(\alpha_1 \, , \beta_1) = \underset{0 \leq d_1 \leq [8/2 \, , \, 8]}{Max} \; \{d_1\}$$

stage 2 :

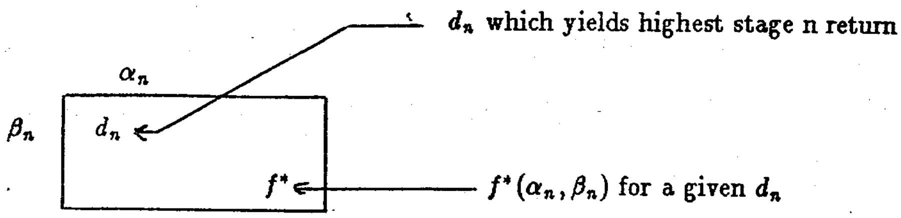
$$f_2^*(\alpha_2\,,\,\beta_2) = \underset{0 \leq d_2 \leq [8\,,\,8/2]}{Max}\ \{d_2 + f_1^*(\alpha_1\,,\,\beta_1)\}$$

where $\alpha_1 = \alpha_2 - d_2\ ,\ \beta_1 = \beta_2 - 2d_2$.

Stage 1 : calculations

β_1 \ α_1	0	1	2	3	4	5	6	7	8
0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0
1	0 0	0 0	1 1	1 1	1 1	1 1	1 1	1 1	1 1
2	0 0	0 0	1 1	1 1	2 2	2 2	2 2	2 2	2 2
3	0 0	0 0	1 1	1 1	2 2	2 2	3 3	3 3	3 3
4	0 0	0 0	1 1	1 1	2 2	2 2	3 3	3 3	4 4
5	0 0	0 0	1 1	1 1	2 2	2 2	3 3	3 3	4 4
6	0 0	0 0	1 1	1 1	2 2	2 2	3 3	3 3	4 4
7	0 0	0 0	1 1	1 1	1 1	2 2	3 3	3 3	4 4
8	0 0	0 0	1 1	1 1	1 1	2 2	3 3	3 3	4 4

Table 1.1



Stage 2 : calculations

d_2	0	1	2	3	4	f_2^*	d_2^*
$\alpha_2 = 8$ $\beta_2 = 8$	4	4	5	5	4	5	2 or 3

Table 1.2

Hence the optimal solution :

$$f_2^*(\alpha_2 = 8\,,\, \beta_2 = 8) \;=\; 5$$

$$d_2^* = x_2^* = 2 \longrightarrow \left.\begin{array}{l} \alpha_1 = \alpha_2 - d_2 = 8 - 2 = 6 \\ \beta_1 = \beta_2 - 2d_2 = 8 - 4 = 4 \end{array}\right\} \longrightarrow x_1^* = d_1^* = 3$$

or

$$d_2^* = x_2^* = 3 \longrightarrow \alpha_1 = 5 \;\&\; \beta_1 = 5 \longrightarrow d_1^* = x_1^* = 2$$

2. An Algorithm for Solving the Knapsack Problem :

Consider the simplest integer program, i.e., an integer program with only one constraint, called the Knapsack problem . We can express the Knapsack problem as follows :

(Kp) : Max

$$\sum_{j=1}^{n} c_j x_j \qquad\qquad (c_j \geq 0\,,\ \ integers)$$

subject to

$$\sum_{j=1}^{n} a_j x_j \leq b \quad (\,a_j \ and\, b \quad +ve\ integers\,)$$

$$x_j \geq 0\ ,\ \text{integer}\ (j = 1, 2, \ldots, n)\ .$$

In order to solve the above problem (Kp), let us define a new function $f(k, \beta)$ to be the maximal value of the objective function using only the first k

($k = 1, 2, ..., n$) items . When the weight limitation is β ($\beta = 0, 1, ..., N$) . That is The problem (IP) can be formulated as a dynamic programming model as follows :

$$\left.\begin{aligned} f(k,\beta) = max \quad & \sum_{j=1}^{k} a_j x_j \\ subject\ to \quad & \sum_{j=1}^{k} a_j x_j \leq \beta \\ & x_j \geq 0\ ,\ \ integer\ (j = 1,\ 2,\ ...,\ k). \end{aligned}\right\} \qquad (1)$$

Isolating x_1 in (1) yields, for $k = 2, ..., n$ and $\beta = 0, 1, ..., N$.

$$\begin{aligned} f(k,\beta) = \max_{x = 0,\ 1,\ ...,\ [\beta/a_k]} \quad & c_k x_k + \sum_{j=1}^{k-1} c_j x_j \\ subject\ to \quad & \sum_{j=1}^{k-1} a_j x_j \leq \beta - a_k x_k \\ & x_j \geq 0\ ,\ \ integer\ (j = 1,\ ...,\ k-1). \end{aligned}$$

$$\begin{aligned} = c_k x_k + \ max \quad & \sum_{j=1}^{k-1} c_j x_j \\ subject\ to \quad & \sum_{j=1}^{k-1} a_j x_j \leq \beta - a_k x_k \\ & x_j \geq 0\ ,\ \ integers\ (j = 1,\ ...,\ k-1). \end{aligned} \qquad (2)$$

By definition, the problem in brackets has an optimal $f(k-1\,,\beta - a_k\,x_k)$. Thus, for $k = 2, ..., n$ and $\beta = 0, 1, ..., N$ (2) can be rewritten as :

$$f(k,\beta) = \max_{x_k = 0,\ 1,\ ...,\ [\beta/a_k]} \quad (c_k x_k + f(k-1\ \beta - a_k\, x_k)) \qquad (3)$$

To start the computations, first we have to find .

$$f(1,\beta) = \max_{0 \leq x_1 \leq [\beta/a_1]} \quad c_1 x_1$$

$$= \begin{cases} 0 & if\ c_1 \leq 0\ \ (where\ x_1 = 0) \\ [\beta/a_1]\,c_1 & if\ c_1 > 0\ \ (where\ x_1 = [\beta/a_1]) \end{cases}$$

and using recursive relation (3) to find $f(2,\beta)$, ..., $f(k,\beta)$, for $\beta = 0, 1, ..., N$, by using the initial condition $f(0,\beta) = 0$ for $\beta = 0, ..., N$ and $f(k,0) = 0$ for $k = 1, ..., n$.

To solve a Knapsack problem with bounded variables and an equality constraint we add the initial conditions $f(0,0) = 0$ and $f(0,\beta) = -\infty$ for $\beta = 1, ..., N$ in place of $f(0,\beta) = 0$. This allow us to use equation (3) directly .

2.1. The Algorithm :

Step 0 : (Initialization) :

Set $f(0,0) = 0$, and $f(k,0) = 0$ for $k = 1, ..., n$.
If constraint of type $\leq$, then set $f(0,\beta) = 0$, $\beta = 1, ..., N$.
Otherwise if constraint of type $=$, then set $f(0,\beta) = -\infty$, $\beta = 1, ..., N$. Go to step 1 .

Step 1 :

Set $k = 1$. If $c_1 \leq 0$ then set $f(1,\beta) = 0$, otherwise set $f(1,\beta) = [\beta/a_1]c_1$ (where $x_1 = [\beta/a_1]$) . Go to step 2 .

Step 2 :

Set $k = k+1$. For all $\beta < a_k$, set $f(k,\beta) = f(k-1,\beta)$. Otherwise compute

$$f(k,\beta) = \max_{x_k = 0, 1, ..., u_k} (c_k x_k + f(k-1, \beta - a_k x_k))$$

(where u_k is an upper bound for x_k) . If $k < n$ then return to step 2 . Otherwise go to step 3 .

Step 3 :

Terminate with optimal integer solution $Z = f(n, N)$.

End (of algorithm) .

Example 2.2 :

Maximize $3\, x_1 + 5\, x_2 + x_3 + x_4$

subject to $2\, x_1 + 4\, x_2 + 3\, x_3 + 2\, x_4 \leq 5,$

$x_1 \leq 1$, $x_2 \leq 1$, $x_3 \leq 1$,

x_1 , x_2 , x_3 , $x_4 \geq 0$, integers .

Using the recursive equation (3) as shown in table 1.

				β					
k	a_k	c_k		0	1	2	3	4	5=
1	2	3	$f(1,\beta)$	0	0	3	3	3	3
			x_1	–	$\boxed{0}$	1	1	1	1
2	4	5	$f(2,\beta)$	0	0	3	3	5	5
			x_2	–	0	0	0	1	$\boxed{1}$
3	3	1	$f(3,\beta)$	0	0	3	3	5	5
			x_3	–	0	0	0	0	$\boxed{0}$
n = 4	2	1	$f(4,\beta)$	0	0	3	3	5	5
			x_4	–	0	0	0	0	$\boxed{0}$

Table 1.3.

The optimal solution is : $f(4,5) = 5$ with $x_4 = 0$ (N is reduced to 5 - 2(0) = 5), $x_3 = 0$ (N is reduced to 5 - 3(0) = 5), $x_2 = 1$ (N is reduced to 5 - 4(1) = 1), and $x_1 = 0$. The optimal values of the variables are boxed .

3. Solving IP Problems by DP Enumeration :

Greenberg (4), has proposed some integer programming methods in a dynamic programming framework . In this section we shall introduce a proposed algorithm for the solution to the IP problem by means of DP enumeration .

First, we find the continous linear programming solution by relaxing the integer restriction on the variables . If the continuous solution is fractional, we develop linear congruences that the nonbasic variables must satisfy, which is added as constraints.

Consider the integer programming problem :

Find integers $x_j \geq 0$ for j = 1, 2, ..., n .

that

$$\begin{array}{ll} minimize & Z = \sum_{j=1}^{n} c_j x_j \\ subject\ to & \sum_{j=1}^{n} a_{ij} x_j = b_i\ ,\ \ i = 1,\ 2,\ ...,\ m \end{array} \tag{4}$$

where a_{ij} , b_i , and c_j are given integer constants .

<u>Remark</u> : we consider the case where some or all of the variables have upper bounds . Thus we include the case where the x_j is 0-1 . Problem (4) may be solved as a LP by relaxing the integer constraints to obtain the optimal canonical form :

$$\left.\begin{array}{ll} min & Z = \overline{Z}_o + \sum_{j \in NB} \overline{c}_j x_j\ , \\ subject\ to & x_B + \sum_{j \in NB} \overline{a}_{ij} x_j = b_i\ ,\ \ i = 1,\ 2,\ ...,\ m \end{array}\right\} \tag{5}$$

where B = the set of indices of the basic variables, NB = the set of indices of the nonbasic variables .

The vectors α_j (j=0 & j $\in$ NB) are column vectors . The components of α_o are nonnegative . If α_o is all ineger, then $x_j = 0$ (j $\in$ NB), $x_B = \alpha_o$ is an optimal solution . Otherwise if any of the x_j (j $\in$ NB) are fractional then the equivalent Knapsack problem is :

$$\begin{array}{ll} minimize & \sum_{j \in NB} c_j x_j \\ subject\ to & \sum_{j \in NB} \beta_j x_j \equiv \beta_o\ \ mod\, 1\ , \\ and & 0 \leq x_j \leq U_j\ ,\ \ x_j\ integer,\ j \in NB\ . \end{array} \tag{6}$$

where β_j are the columns of fractional parts of the α_j from (5) . Now, to solve the problem (6), we form the Knapsack function :

$$F(\beta) = min \left[\sum_{j \in NB} c_j x_j \mid \sum_{j \in NB} \beta_j x_j \equiv \beta \; mod\, 1, \; x_j \leq U_j \right] \tag{17}$$

which may be written as the dynamic programming recursion

$$F(\beta) = \min_j \; [c_j + F(\beta - \beta_j)] \tag{18}$$

where the arguments of F takes modulo 1 . The recursion in (8) may be solved as a simple enumeration by noticing that :

$$F(\beta_r) = c_r \tag{19}$$

where

$$c_r = \min_{j \in NB} c_j \tag{20}$$

Then from $F(\beta - \beta_r)$ by replacing β by $\beta - \beta_r$ in (8) and we substitute the result for the $F(\beta - \beta_r)$ term on the R.H.S. of (8) . As

$$F(\beta - \beta_r) = \min_j [c_j + F(\beta - \beta_j - \beta_r)]$$

we then can produce another immediate solution, keeping the upper bounds U are not violated while performing the enumeration . To solve the integer program (5) we stop the enumeration when $F(\beta_o)$ is calculated .

3.1. The Algorithm :

Step 1 :

Suppose that the set of non-basic indices NB = (1, 2, ..., m) . Then list the values of the program as :

1	2	3	. . .	m
c_1	c_2	c_3	. . .	c_m
β_1	β_2	β_3	. . .	β_m
$x_j = 0\ ,\ j = 0, 1, ..., m$				

Table 1.4.

Go to step 2 .

Step 2 :

Given the list, find $c_r = \min c_j$ for all unmarked columns in all sections. If $\beta_r = \beta_o$ then mark the column . Go to step 4 . Otherwise mark the column. Go to step 3 .

Step 3 :

Add a new section of columns to the list, if possible as follows :

(i) Calculate $c'_j = c_r + c_j\ ,\ \beta'_j \equiv \beta_r + \beta_j \pmod 1\ \ \forall j \in NB$.

(i.e. the values c_j & β_j are taken from the original list in step 1) . Where $x_j < U_j$ for $j \neq r$ and

$$x_{r+1} + 1 < U_r \quad for j = r$$

for the section containing the newly marked column .

(ii) Add the columns labelled by j in the new section with values c'_j and β'_j .

(iii) Under the section added write the x_j values from the section containing the newly marked column . Set $x_r = x_r + 1$ for the section . Go to step 2 .

Step 4 :

Take as a trial solution the values of the variables found below the section where $\beta_r = \beta_o$ appears with x_r increased by one .

If the constraints in (5) are satisfied then the solution found is the optimal integer solution to the original integer program (4) . Otherwise go to step 3 .

End (of algorithm) .

Example 3.2 :

$$\begin{aligned}
\text{Minimize} \quad & 5x_1 + 7x_2 + 10x_3 + 3x_4 + x_5 \\
\text{subject to} \quad & x_1 - 3x_2 + 5x_3 + x_4 - 4x_5 \geq 2 \\
& -2x_1 + 6x_2 - 3x_3 - 2x_4 + 2x_5 \geq 0 \\
& + x_2 + 2x_3 - x_4 - x_5 \geq 1
\end{aligned}$$

and $0 \leq x_j \leq 1$, x_j integer , j=1, ..., 5.

Using the Lexic. dual simplex method to find the continuous solution, we have the equivalent problem :

$$\begin{aligned}
\text{Min} \quad & 9 + \tfrac{93}{9}x_1 + \tfrac{158}{9}x_4 + \tfrac{42}{9}x_5 + \tfrac{24}{9}x_7 + \tfrac{81}{9}x_8 \\
\text{subject to} \quad & -\tfrac{7}{9}x_1 - \tfrac{28}{9}x_4 + \tfrac{13}{9}x_5 + x_6 + \tfrac{1}{9}x_7 - \tfrac{14}{9}x_8 = \tfrac{3}{9} \\
& -\tfrac{2}{9}x_1 + x_3 - \tfrac{8}{9}x_4 - \tfrac{4}{9}x_5 - \tfrac{1}{9}x_7 - \tfrac{6}{9}x_8 = \tfrac{6}{9} \\
& -\tfrac{4}{9}x_1 + x_2 - \tfrac{7}{9}x_4 + \tfrac{1}{9}x_5 - \tfrac{2}{9}x_7 - \tfrac{3}{9}x_8 = \tfrac{3}{9}
\end{aligned}$$

and $0 \leq x_j \leq 1$, x_j integer , j=1, ..., 5.

$x_j \geq 0$, j=6, 7, 8.

and develop the congruences

$$\begin{aligned}
\tfrac{2}{9}x_1 + \tfrac{8}{9}x_4 + \tfrac{4}{9}x_5 + \tfrac{1}{9}x_7 + \tfrac{6}{9}x_8 &\equiv \tfrac{3}{9} \bmod 1 \\
\tfrac{7}{9}x_1 + \tfrac{1}{9}x_4 + \tfrac{5}{9}x_5 + \tfrac{8}{9}x_7 + \tfrac{3}{9}x_8 &\equiv \tfrac{6}{9} \bmod 1 \\
\tfrac{5}{9}x_1 + \tfrac{2}{9}x_4 + \tfrac{1}{9}x_5 + \tfrac{7}{9}x_7 + \tfrac{6}{9}x_8 &\equiv \tfrac{3}{9} \bmod 1
\end{aligned}$$

Step 1 : The problem is listed in tableau T1; $\beta = (3/9, 6/9, 3/9)$, $z_o = 9$.

Step 2 : r = 7 in tableau T1, $c_r = 24/9$.

j	1	4	5*	7*	8	1	4	5*	7*	8
c_j	$\frac{93}{9}$	$\frac{156}{9}$	$\frac{42}{9}$	$\frac{24}{9}$	$\frac{81}{9}$	$\frac{117}{9}$	$\frac{180}{9}$	$\frac{66}{9}$	$\frac{48}{9}$	$\frac{105}{9}$
	2/9	8/9	4/9	1/9	6/9	3/9	0	5/9	2/9	7/9
β_j	7/9	1/9	5/9	8/9	3/9	6/9	0	4/9	7/9	2/9
	5/9	2/9	1/9	7/9	6/9	3/9	0	8/9	5/9	4/9
	$x_j = 0$					$x_7 = 1$				

T1 T2

1	4	8	1	4	5	7	8	1	4	8
$\frac{135}{9}$	$\frac{198}{9}$	$\frac{123}{9}$	$\frac{141}{9}$	$\frac{204}{9}$	$\frac{90}{9}$	$\frac{72}{9}$	$\frac{129}{9}$	$\frac{159}{9}$	$\frac{222}{9}$	$\frac{147}{9}$
6/9	3/9	1/9	4/9	1/9	6/9	3/9	8/9	7/9	4/9	2/9
3/9	6/9	8/9	5/9	8/9	3/9	6/9	1/9	2/9	5/9	7/9
6/9	3/9	7/9	1/9	7/9	6/9	3/9	2/9	4/9	1/9	5/9
$x_5 = 1$			$x_7 = 1$					$x_5 = 1$, $x_7 = 1$		

T3 T4 T5

<u>Step 3</u> : We form tableau T2. Mark column 7 of tableau T1.

<u>Step 2</u> : r = 5 in tableau T1, $c_r = 42/9$.

<u>Step 3</u> : We form tableau T3. Mark column 5 of tableau T1. We need not add a column headed by 7 in T3 because it would duplicate the 5 column in T2. Also we do not add a 5 column in T3 since x_5 is at its upper bound in T3.

<u>Step 2</u> : r = 7 in tableau T2, $c_r = 48/9$.

<u>Step 3</u> : We form tableau T4. Mark column 7 of tableau T2.

<u>Step 2</u> : r = 5 in tableau T2, $c_r = 66/9$.

<u>Step 3</u> : We form tableau T5. Mark column 5 of tableau T2. The solution is now possible in the 7 column in T4. We have $x_7 = 3$, $x_1 = x_4 = x_5 = x_8 = 0$. Substitute $x_7 = 3$ into constraint equations, we obtain $x_6 = 0$, $x_8 = 1$, and $x_2 = 1$.

Thus we have achived the optimal solution with objective value $9 + \frac{72}{9} = 17$. Note that if x_6 , x_3 , or x_2 are not feasible, other solutions are possible; e.g., the 1 column of (T2), the 4 column of (T3), and others if the enumeration is continued .

4. Conclusions :

In section 1, formulation the IP as a DP model is introduced . Unfortunately, the presence of multiple state variables creates major difficulty in the solution of DP problems from computational viewpoint . This is called the curse of dimensionality in DP . Thus for large problems this is not recommended as a general approach .

In section 2, we presented algorithm for solving the Knapsack problem. For small problems (when the numbers of variables, and the constants are relatively small) the DP approach performs well . The problem constants must be +ve, and integers, the variables must be bounded also . For variable x_j without upper bound, we simply solve a linear program : maximize x_j subject to the constraints of the original problem . The -ve coefficient variable x_j may be transformed to a +ve coefficient, by setting $x_j = U_j - x_j$ (where U_j is the upper bound for the variable x_j) .

The algorithm discussed in section 3, is proposed by *Greenberg* [3] . Indeed, this is an enumerative method in a DP framework . The method can be used to solve the important class of problems in which the variables have upper values .

From the foregoing discussion, we can deduce that, the DP technique is not recommended as a general approach for solving the IP problems, since it suffers from dimensionality problem .

Solving Integer Programming.

Although, the branch-and-bound methods have the advantage of solving both the IP and MIP [2], however, it requires extremely large computer storage capacity . For problems with many variables . Thus the mixture of the branch-and-bound and the DP may have a double advantage . The first, is the generality of branch-and-bound in solving the IP and MIP problems . The second, is the advantage of utilizing DP from computation efficiency viewpoint . For example, the DP technique may be used in branch and bound for computation of bounds, also it may be used for fathoming critera .

REFERENCES

1. Glover, F., "New Results for Reducing Linear Programs to Knapsack Problems", Management Science Report, Series 72-7, University of Colorado (1972).

2. Gomory, F., "On the relation between Integer and Non-Integer solutions to Linear Programs", Proc. Mat. Acad. Sci., 260–265 (1965).

3. Greenberg, H., "A Dynamic Programming Solution to Integer Linear Programs", J. Math. Anal. Appl., 26 (2), 454–459, (1969).

4. ————, "Integer Programming", Academic Press (1971).

5. Hadly, "Nonlinear and Dynamic Programming", Reading Mass., Addison-Wesley, (1964).

6. Jeromin, B., and F. Korner, "A Hybrid Method For Solving Integer Linear Programming Problems", Matematicky Obzor, 22 (4), 400–407 (1986).

[2]MIP = Mixed Integer Programming

7. Kendel, K., and S. Zionts, "Solving Integer Programming Problems by Aggregating Constraints", School of Management, Working Paper No. 155, November (1972).

8. Kovacs, L., "Solution of Linear Integer Programming Problems by Dynamic Programming", Math. Operations for Sch. U. Statist. 5, Heft 3, 163–176, (1974).

9. Morin, T., and R. Marsten, "Branch-and-Bound Strategies for Dynamic Programming", Operations Research, Vol. 24, No. 4 (1976).

10. Salkin, H.M., "Integer Programming", Addison-Wesley, (1975).

11. Shapiro, J.F., "Dynamic Programming Algorithms for the Integer Programming Problem - I : The Integer Programming Problem viewed as a Knapsack type Problem", Opers. Res. 16, 103–121 (1968).

ايجاد حل مشكلة البرمجة الخطية بإستخدام البرمجة الديناميكية

قدم كثير من الباحثين أمثال جلوفر وكندل وزيونتس منهاج لتحويل مشكلة البرمجة الصحيحة الأصلية الى مشكلة نابساك Knapsack بإعتبار أن حل هذه المسألة يكون أسهل بإستخدام توالى البرمجة الديناميكية.

وقد حول بحاث آخرون مثل جومورى وشابيرو مسألة البرمجة الصحيحة الى مسألة مجموعة أمثلية والتى تشابه بشدة مسألة نابساك والحل الأمثل للمسألة المحولة غالباً مايعطى حلاً أمثل لمسألة البرمجة الصحيحة الأصلية المستنتجة منها.

وهنا فى هذا البحث قد طورنا طريقة أخرى وذلك بدمج طرق البرمجة الديناميكية مع نظرية الفرع والحد branch and bound principle وقد آثبت مورين ومارستن مدى امكانية إستخدام طرق الفرع والحد لتقليص التخزين بل وممكن أيضاً المتطلبات الحسابية فى برامج الديناميكية غير الصحيحة.

والهدف من هذا البحث هو اثبات مدى امكانية استخدام البرمجة الديناميكية DP فى حل مسائل البرمجة الصحيحة IP .