Multicriteria Dynamic Programming with an Application to the Integer Case

B. Villarreal and M. H. Karwan

Communicated by P. L. Yu

Abstract. Fundamental dynamic programming recursive equations are extended to the multicriteria framework. In particular, a more detailed procedure for a general recursive solution scheme for the multicriteria discrete mathematical programming problem is developed. Definitions of lower and upper bounds are offered for the multicriteria case and are incorporated into the recursive equations to aid problem solution by eliminating inefficient subpolicies. Computational results are reported for a set of 0-1 integer linear programming problems.

Key Words. Multicriteria optimization, dynamic programming, discrete optimization, integer programming.

1. Introduction

In a complex organizational environment, the need to take into account the different effects of a strategic decision over various objectives or performance criteria has been recognized by many researchers. As a result, important decision processes are often formulated in a multicriteria framework [see, for example, Krouse (Ref. 1) and Rueffi (Ref. 2)]. Various approaches have emerged that provide solutions to a wide variety of multicriteria problems. One approach is to generate the set of efficient or Pareto-optimal points or decisions [Geoffrion (Ref. 3) Bitran (Ref. 4), Yu and Zeleny (Ref. 5)]. Others have developed interactive approaches to

---

1 This research was supported in part by CONACYT (Consejo Nacional de Ciencia y Tecnología), Mexico City, Mexico.
2 Instituto Tecnologico y de Estudios Superiores de Monterrey, Monterrey, Nuevo Leon, Mexico.
3 Associate Professor, Department of Industrial Engineering, State University of New York at Buffalo, Buffalo, New York.
solve multicriteria problems [Zionts and Wallenius (Ref. 6), Geoffrion, Dyer, and Feinberg (Ref. 7), Benayoum, Montgolfier, Tergny, and Laritchev (Ref. 8), Chankong and Haimes (Ref. 9)]. Finally, Lee (Refs. 10 and 11) and Lee and Morris (Ref. 12), among others approach the solution by setting preassigned goal levels or targets for the relevant criteria of the problem of concern.

An important decision process that has been analyzed in depth for certain specific conditions is the serial multistage decision process. A graphical representation of the process is shown in Fig. 1, where the following notation is employed for \( n = 1, \ldots, N \):

- \( x_n \triangleq n\text{th decision variable}; \)
- \( Y_n = (y_{1n}, \ldots, y_{Mn})^T \triangleq n\text{th vector of state variables}; \)
- \( S_n(Y_n) \triangleq \text{set of alternatives when state } Y_n \text{ is attained}; \)
- \( r_n(x_n, Y_n, Y_{n-1}) \triangleq n\text{th return function of inputs, outputs, and decisions}; \)
- \( t_n(x_n, Y_n) = (t_{1n}(x_n, y_{1n}), \ldots, t_{Mn}(x_n, y_{Mn}))^T \triangleq n\text{th transformation vector composed of single-valued transformation functions.} \)

At a given stage, inputs are received, and a transformation process acts upon these inputs and the decision is made. This results in an output to be used at the following stage and a return at the present stage (see Fig. 1). A problem of great interest that is related to this process is one in which the total return of the decision process is required to be maximized, for given initial state vectors of inputs \( Y_n \), subject to each of the transformation functions at each stage. After taking advantage of the type of transformation functions at hand, Nemhauser (Ref. 13) offers the following mathematical formulation of this problem:

\[
\begin{align*}
\max_{x_1, \ldots, x_N} & \quad g\{r_1^0(x_1, Y_1), \ldots, r_N^0(x_N, Y_N)\}, \\
\text{s.t.} & \quad Y_{n-1} = t_n(x_n, Y_n), n = 1, \ldots, N,
\end{align*}
\]

where \( g\{r_1^0(x_1, Y_1), \ldots, r_N^0(x_N, Y_N)\} \) denotes an arbitrary function of the return functions of each stage. The solution to this problem is an optimal policy \((x_1, \ldots, x_N)\) that maximizes the total return of the decision process for a given initial vector of inputs \( Y_N \). Mitten (Ref. 14) suggests sufficient conditions that the total return function \( g\{r_1^0(x_1, Y_1), \ldots, r_N^0(x_N, Y_N)\} \) must satisfy in order to solve the problem via a dynamic programming recursive approach.

---

4 Mathematical rigor requires that \( r_n(x_n, Y_n, Y_{n-1}) \), a function with three arguments, be distinguished from a similar function with two arguments. Thus, we employ \( r_n^0(x_n, Y_n) \), although the conventional practice in dynamic programming is to use the same function \( r \), with different numbers of arguments.
An important extension of this problem would be that of allowing for the evaluation of the decisions made at each stage in a multicriteria framework. Thus, one would be interested in finding a set of policies \( \{(x_1, \ldots, x_N)\} \), in terms of various criteria, instead of achieving an optimal policy in terms of only a single criterion or objective. To allow for this extension, it is necessary to have a finite return vector

\[
R^n(x_n, Y_n) = (r_{1n}(x_n, Y_n), \ldots, r_{pn}(x_n, Y_n))
\]

at each stage of the process. Then, the serial multicriteria multistage problem (SMMP) would be formulated as follows:

\[
\text{(SMMP) \quad } \max_{x_1, \ldots, x_N} G\{R^1(x_1, Y_1), \ldots, R^N(x_N, Y_N)\},
\]

s.t. \( Y_{n-1} = t_n(x_n, Y_n), n = 1, \ldots, N \),

where \( G\{R^1(x_1, Y_1), \ldots, R^N(x_N, Y_N)\} \) denotes the multicriteria return function of the decision process and \( \max \) (vector maximization) is used to differentiate the problem from the maximization problem. The solution to this problem would be a set of Pareto-optimal policies

\[
x^0 = (x_1^0, \ldots, x_N^0) \in X^0(Y_N),
\]

with the property that, for a given initial vector of inputs \( Y_N \), there does not exist a feasible policy \( (x_1, \ldots, x_N) \), such that

\[
G\{R^1(x_1, Y_1), \ldots, R^N(x_N, Y_N)\} \geq G\{R^1(x_1^0, Y_1), \ldots, R^N(x_N^0, Y_N)\},
\]

with at least one strict inequality.

It would be desirable to decompose this problem similarly to the way in which the single-objective maximization problems have been decomposed for special conditions of the total return functions and transformation functions [see, for example, Bellman and Dreyfus (Ref. 15), Mitten (Ref.
14), Nemhauser (Ref. 13)]. These single-objective problems are decomposed into equivalent problems that involve the solution to $N$ maximization subproblems, each containing only one vector of state variables (inputs) $Y_n$ and one decision variable $x_n$.

Mitten (Ref. 16) provides some theory and an untested algorithm for the multicriteria problem under a structure of preference relations adapted to an interactive mode of implementation in which there is a dialogue between the decision maker and a source of information and analysis. Villarreal (Ref. 17) shows that this approach is impractical and develops an imbedded-state interactive procedure. However, computer time and storage requirements still remain prohibitive for large problems. Brown and Strauch (Ref. 18) develop an analogue of the principle of optimality under a setting provided by the concept of conditionally complete multiplicative lattices. Whereas many of their results are of the same flavor as some given here, they cannot be directly applied to the problem considered in this paper.

The plan of this paper is as follows. Section 2 presents some basic results and definitions required for the development of more relevant theory in later sections. The extension of dynamic programming recursive equations to the multicriteria framework is reported in Section 3. These results are also reported in Yu (Ref. 19) in a more general case of domination structures and discussed here briefly. Section 4 is concerned with developing a general procedure for multicriteria discrete programming. Section 5 provides definitions of lower and upper bounds in the multicriteria case which are later used to greatly reduce computation time for selected test problems. Computational results are given in Section 6 for a set of 0–1 integer linear programming problems.

2. Basic Results and Definitions

Nemhauser (Ref. 13) employs the assumptions that the transformation functions $t_n(x_n, Y_n)$ are single valued, to express the return function for stage $n$, $r_n(x_n, Y_n, Y_{n-1})$, by an equivalent expression $r_n^0(x_n, Y_n)$. This assumption ensures that, for the same given values of $x_n$ and $y_{in}$, one and only one unique value of $y_{in-1}$ will be obtained. Thus,

$$r_n(x_n, Y_n, Y_{n-1}) = r_n^0(x_n, Y_n, t_n(x_n, Y_n)) = r_n^0(x_n, Y_n).$$

Due to the structure of the decision process, some important assumptions are implied. One is that the vector of outputs of the $n$th stage $Y_{n-1}$ depends only upon the amount of inputs available at the start of the process
It then follows that
\[ r_n^o(x_n, Y_n) = \beta_n(x_n, x_{n+1}, Y_{n+1}) = \cdots = \beta(x_n, \ldots, x_N, Y_N). \]

Using these assumptions, the serial multicriteria multistage problem (SMMP) is defined as a vector-maximization problem over the decision variables \( x_n, n = 1, \ldots, N \), and an initial vector of inputs \( Y_N \). To facilitate the development of recursive equations for (SMMP), new notation and definitions are introduced as they are required.

Let \( \Delta \) denote a \( p \)-dimensional vector of binary operators, which are to be applied to elements corresponding to the same components of two vectors. For example, if
\[ a = (1, 3, 4)^t, \quad b = (4, 6, 8)^t, \quad \Delta = (+, /, \cdot)^t, \]
then
\[ a \Delta b = (1 + 4, 3/6, 4 \cdot 8) = (5, 1/2, 32). \]

Now, consider the following definition, given in Villarreal and Karwan (Ref. 20), which is comparable to Yu’s definition of separability (Ref. 19).

**Definition 2.1.** A multicriteria return function \( R^1(x_1, Y_1) \Delta \cdots \Delta R^n(x_n, Y_n) \) is said to be stagewise separable if it can be reconstructed by the iterative use of the corresponding vector of operators \( \Delta \).

An example of the above definition is the function
\[ R_4(x_4, Y_4) + R_3(x_3, Y_3) + R_2(x_2, Y_2) \cdot R_1(x_1, Y_1). \]

It can be reconstructed stagewise as follows. Let
\[ R_1 = R^1(x_1, Y_1), \quad \text{at stage } n = 1. \]

At stage \( n = 2 \), one has that
\[ R_2 = \{ R^2(x_2, Y_2) \cdot R_1 \}. \]

At stage \( n = 3 \),
\[ R_3 = \{ R^3(x_3, Y_3) + R_2 \}. \]

Finally, at \( n = 4 \),
\[ R_4 = \{ R^4(x_4, Y_4) + R_3 \}. \]

Substituting back, one can observe that \( R_4 \) is, in fact,
\[ R_4 = R^4(x_4, Y_4) + R^3(x_3, Y_3) + \{ R^2(x_2, Y_2) \cdot R^1(x_1, Y_1) \}. \]
On the other hand, observe that the function
\[ R^4(x_4, Y_4) \cdot R^3(x_3, Y_3) + R^2(x_2, Y_2) + R^1(x_1, Y_1) \]
is not stagewise separable in the same sense as the first example above. However, if one regards
\[ \{ R^4(x_4, Y_4) \cdot R^3(x_3, Y_3) \} \]
as one combined function corresponding to a stage, one will be able to reconstruct the original total function. Hence, in the first example, the function may be regarded as completely stagewise separable. The degree of separability will depend on the number of individual terms or stages that cannot be totally separated. This paper will not be concerned with the measure of separability, but will assume that the total multicriteria function, considered in what follows, is completely stagewise separable. Notice that, in the examples given above, the vector of binary composition operators \( \Delta \) consists of the same operator for all the elements of the same stage. However, this is not required. Also, as in the above examples the vector of operators \( \Delta \) can vary in its composition at each stage. If the operators of each stage are different, the condition of complete separability must apply to each individual criterion function of the set. If only one of the criterion functions is not completely stagewise separable, the total function will not be completely stagewise separable. For example, the multicriteria functions
\[ \sum_{j=1}^{n} R^j(x_j, Y_j) \quad \text{and} \quad \prod_{j=1}^{n} R^j(x_j, Y_j) \]
are easily seen to be completely stagewise separable.

Let us now consider the following definitions [amended from Yu (Ref. 19)], which will limit the type of operators \( \Delta \) permissible for a recursive approach to (SMMP) to guarantee success.

Recall that \( S_n(Y_n) \) denotes the set of alternatives for \( x_n \) when state \( Y_n \) is reached.

**Definition 2.2.** The attainable return set from \( Y_n, n = 1, \ldots, N, \) is denoted by \( \text{AR}(Y_n) \), where
\[ \text{AR}(Y_n) = \{ R^n(x_n, Y_n) | x_n \in S_n(Y_n) \}. \]

Note that
\[ \text{AR}(Y_k) = \cup \{ R^k(x_k, Y_k) \Delta \text{AR}(t_{k-1}(x_k, Y_k)) | x_k \in S^k(Y_k) \}, \]
for \( k = 2, \ldots, N, \)
where

\[ R^k(x_k, Y_k) \Delta \mathcal{A} \mathcal{R}(Y_{k-1}) = \{ R^k(x_k, Y_k) \Delta z_{k-1} | z_{k-1} \in \mathcal{A} \mathcal{R}(Y_{k-1}) \}. \]

**Definition 2.3.** The multicriteria return function is monotonic if and only if, for all \( z_{k-1}, z'_{k-1} \in \mathcal{A} \mathcal{R}(i_{k-1}(x_k, Y_k)) \), such that \( z_{k-1} > z'_{k-1} \), we have

\[ R^k(x_k, Y_k) \Delta z_{k-1} > R^k(x_k, Y_k) \Delta z'_{k-1}. \]

For example, the vector of operators

\[ \Delta = (\Delta_1, \Delta_2, \ldots, \Delta_p), \]

where \( \Delta_i = + \) for all \( i \), will result in a monotonic return function over the real line. Also, the vector of operators \( \Delta \), composed of only product operators will produce a monotonic return function over the range of nonnegative values. Observe that a vector of operators \( \Delta \) that is composed of various addition operators and product operators will also give a monotonic return function over the range of nonnegative values. Definition 2.3 is an extension of the requirements of a return function to be optimized via dynamic programming, given by Mitten (Ref. 14).

3. **Extension of Dynamic Programming Recursions**

Let the multicriteria function of the form

\[ G\{R^1(x_1, Y_1), \ldots, R^N(x_N, Y_N)\} = R^N(x_N, Y_N) \Delta \cdots \Delta R^1(x_1, Y_1), \]

be completely stagewise separable and monotonic. Thus, (SMMP) can be reformulated as follows:

\[ \text{V-max}\{ R^N(x_N, Y_N) \Delta R^{N-1}(x_{N-1}, Y_{N-1}) \Delta \cdots \Delta R^1(x_1, Y_1) \} \]

s.t. \( Y_{n-1} = t_n(x_n, Y_n) \), \( n = 1, \ldots, N \).

Villatorreal and Karwan (Ref. 20) and Yu (Ref. 19) show that the problem (SMMP) can be solved using the following recursive equations (DSMMP) for \( n = 1, 2, \ldots, N \):

\[ \text{DSMMP} \quad H_n(Y_n) = \text{V-max}_{x_n}\{ R^n(x_n, Y_n) \Delta \mathcal{H}_{n-1}(Y_{n-1}) \}, \]

s.t. \( Y_{n-1} = t_n(x_n, Y_n) \),

where \( H_0(Y_0) \) is defined such that, if there is another finite \( p \)-dimensional
vector $F$, then

$$F \triangleleft H_0(Y_0) = F.$$  

The symbol $\triangleleft$ means that the vector of operators $\Delta$ is to be performed over all the components of $H_{n-1}(Y_{n-1})$. For example, let

$$R'(x_n, Y_n) = (2, 3, 4)^t,$$
$$H_{n-1}(Y_{n-1}) = \{(2, 2, 2)^t, (4, 4, 4)^t\},$$
$$\Delta = (+, \cdot, /).$$

Then,

$$R'(x_n, Y_n) \Delta H_{n-1}(Y_{n-1}) = \{(4, 6, 2)^t, (6, 12, 1)^t\}.$$  

It can easily be shown that a necessary condition for a Pareto-optimal point is that it must contain, as its first $m-1$ components, an efficient solution to an $(m-1)$-stage problem. Also, the conditions of complete stagewise separability and monotonicity of the return function are sufficient to guarantee that each efficient point be generated by the recursive equations. Yu (Ref. 19) reports more general results for domination structures under which the concept of Pareto optimality is a special case.

An important application [see Bitran (Refs. 4 and 21) and Zionts (Refs. 22 and 23)] of the above result is the multicriteria discrete programming problem to be discussed in the next section.

4. Multicriteria Discrete Mathematical Programming Problem

Consider the multicriteria discrete mathematical programming problem (MDMP) defined as follows:

$$(\text{MDMP}) \quad \text{V-max}\{R^N(x_N) \Delta \cdots \Delta R^1(x_1)\},$$

$$\sum_{n=1}^{N} A^n(x_n) \leq b,$$
$$s.t. \quad S_N \triangleq \begin{cases} k_n \geq x_n \geq 0, \text{ integer, } & n = 1, \ldots, N, \\
(b_i \geq 0, & i = 1, \ldots, M.\end{cases}$$

Here, the return function is completely stagewise separable; each function $r_{in}(x_{in})$ is an arbitrary finite function of $x_n$; $A^n = (a_{1n}, a_{2n}, \ldots, a_{Mn})$; each function $a_{in}(x_{in})$ is an arbitrary finite single-valued function of $x_n$; $k_n$ is a nonnegative integer constant; $\Delta$ is a vector of operators that can vary.
among columns (stages) and within columns (criteria); and V-max (vector maximization) is used to differentiate the solution from the maximization of a single objective function. It is assumed that the return function is monotonic.

For the case in which
\[ r_{in}(x_n) \geq 0 \quad \text{and} \quad a_{in}(x_n) \geq 0, \]
the problem can be called the multicriteria nonlinear multidimensional knapsack problem. For the case in which
\[ x_n = 0 \text{ or } 1, \quad n = 1, \ldots, N, \]
one would have the multicriteria nonlinear zero–one integer programming problem. Marsten and Morin (Ref. 24) have developed a methodology to solve the discrete mathematical programming problem for the single objective case for the vector of operators composed only of addition operators. Morin and Marsten (Ref. 25) also suggest a procedure to be used for solving the separable nonlinear multidimensional knapsack problem.

In order to solve (MDMP) by a recursive dynamic programming approach, the constraint set \( S_N \) must be replaced by an equivalent constraint set, denoted by \( S_D \), that is amenable for a dynamic programming formulation. The appropriate set of constraints \( S_D \) is given by [see, for example, Nemhauser (Ref. 13) and Bellman and Dreyfus (Ref. 15)]
\[
S_D^N \triangleq \begin{cases} 
Y_{n-1} = Y_n - A^n(x_n), & \text{for } n = 2, \ldots, N, \\
Y_0 = Y_1 - A^1(x_1), & \\
Y_N \leq b \ (\geq 0), & \\
k_n \geq x_n \geq 0, \text{ integer, } & \text{for } n = 1, \ldots, N.
\end{cases}
\]

Thus, one can use the following recursive equations to obtain the set of efficient solutions:

(DMDMP) \[ H_n(Y_n) = \text{V-max}\{R^n(x_n) \otimes H_{n-1}(Y_n - A^n(x_n))\}, \quad n = 1, \ldots, N, \]
s.t. \[ A^n(x_n) \leq Y_n, \ k_n \geq x_n \geq 0, \text{ integer}, \]

where \( H_n(Y_n) \) is defined as in Section 3.

One can observe that using the above recursive equations will imply solving each \( n \)-stage subproblem for each possible vector of values \( Y_n \). This requirement is obviously necessary, since at the last stage one needs the efficient set of solutions for each vector of resources remaining after making each decision (vary the value of \( x_N \)).

This implication leads to the main difficulty found in the solution methodologies based on these type of recursions. The difficulty has been
called the *curse of dimensionality*, which increases in degree as the number of state variables or constraints at each stage increases [see, for example, Bellman and Dreyfus (Ref. 15), Morin and Marsten (Ref. 25), Nemhauser (Ref. 13)]. Morin and Esoboque (Ref. 26) have developed other recursions, called imbedded-state recursive equations. Using these recursions enables one to overcome many of the problems caused by the curse of dimensionality. However, these recursions have only been developed to solve the maximization of single objective problems. The following discussion extends such equations to the multicriteria discrete mathematical programming problem. Consider the problem (MDMP), and let $A^m(x_n)$ be a vector of continuous functions such that:

(i) $a_{in}(0) = 0$;

and, for all $x_n \geq 0$, either

(ii) $a_{in}(x_n) \geq 0$

and is a monotonically increasing function of $x_n$; or

(iii) $a_{in}(x_n) \leq 0$

and is a monotonically decreasing function of $x_n$.

The range of values over which the vectors $Y_n$ must vary depends on the type of functions $A^m(x_m)$, $m = 1, \ldots, N$. If all $A^m(x_m) \geq \{0\}$, then the range is

$$\{0\} \leq Y_n \leq b.$$  

For the case in which all $A^m(x_m) \leq \{0\}$, the range is given by

$$b \leq Y_n \leq \left\{ b + \sum_{m=1}^{N} |A^m(k_m)| \right\}.$$  

In general, one will have the following two subsets of functions of $x_m$:

$$G_m = \{i|a_{im}(x_m) \geq 0\} \quad \text{and} \quad H_m = \{j|a_{jm}(x_m) \leq 0\}.$$  

In this case, the range for each component $y_{in}$ is given by

$$0 \leq y_{in} \leq \left\{ b_{l} + \sum_{m=1}^{N} |a_{im}(k_m)| \right\} = u_{in}.$$ (1)
The set of constraints $\bar{S}_n$ can be equivalently enforced using the following constraint on $x_n$:

$$\bar{S}_n \Delta \left\{ \text{integer } x_n \begin{cases} 0 \leq x_n \leq k_n, & \text{if } a_{in}(x_n) \leq 0, \text{ for all } i \text{ and } x_n \\ 0 \leq x_n \leq \min_{i \in G_n}(k_n, [a_{in}^{-1}(y_{in})]), & \text{otherwise} \end{cases} \right\},$$

where

$$G_n = \{i \mid a_{in}(x_n) \geq 0\},$$

$[p]$ denotes the greatest integer part of $p$, and $a_{in}^{-1}(y_{in})$ is finite over the range of $y_{in}$ given by (1).

A general scheme based upon the recursions (DMDMP) requires, as a first step, the variation of the values of the vectors $Y_n$. However, this approach leads to the problem of dimensionality. It would be of interest to analyze the behaviour of the solutions $H_n(Y_n)$ with respect to variations in the values of the vectors $Y_n$ for developing a more efficient computational procedure. This analysis will be undertaken in a stagewise fashion for a clearer exposition of these properties.

Let $n = 1$. Then,

$$H_1(Y_1) = \text{V}-\text{max}\{R^1(x_1) \Delta H_0(Y_1 - A^1(x_1))\},$$

s.t. $x_1 \in \bar{S}_1 = \left\{ \text{integer } x_1 \begin{cases} 0 \leq x_1 \leq k_1, & \text{if } A^1(x_1) \leq \{0\}, \text{ for all } x_1 \\ 0 \leq x_1 \leq \min_{i \in G_1}(k_1, [a_{i1}^{-1}(y_{i1})]), & \text{otherwise} \end{cases} \right\},$$

where

$$G_1 = \{i \mid a_{i1}(x_1) \geq 0\}.$$

As a result of the definition of $H_0(Y_0)$, the recursions reduce to

$$H_1(Y_1) = \text{V}-\text{max}\{R^1(x_1)\},$$

s.t. $x_1 \in \bar{S}_1$.

Let us assume that some

$$a_{i1}(x_1) \geq 0, \quad \text{for all } x_1.$$

Thus, the one-stage problem reduces to a vector-maximization problem of functional values resulting from varying the value of $x_1$ over the range

$$[0, \min(k_1, [a_{i1}^{-1}(y_{i1})] \mid i \in G_1)].$$

The upper limit of the range is defined by either the upper bound $k_1$ or the values of the vector $Y_1$. Note that the number of functional values to
be considered in the problem remains constant or increases proportionally with the increases in the values of the vector \( Y_1 \). It will remain constant if the increase in the values of the vector \( Y_1 \) is not enough to cause an increase in the value of \( x_1 \). It will also remain constant after

\[
Y_{i_1} = a_{i_1}(k_1), \quad \text{for any } i \in G,
\]

since further increases in the values of the vector \( Y_1 \) do not increase the range, which is limited by \( k_1 \). An obvious result of this analysis [see Villarreal (Ref. 12)] is the following.

**Lemma 4.1.** The solution set as a function of \( Y_1, H_1(Y_1) \), varies in a stepwise manner, due to corresponding changes in the value of \( x_1 \).

During the procedure for solving the one-stage problem for \( n = 1 \), the set of efficient solutions for each vector of values \( Y_1 \) will either remain constant or change. The changes in these sets correspond to variations in the values of \( x_1 \). This result enables one to modify a portion of the general scheme suggested by the recursions as follows.

**Step 1.** Consider the range of values of \( x_1 \), \((0, 1, \ldots, k_1)\).

**Step 2.** Eliminate those values of \( x_1 \) such that any component of its resource consumption vector \( a_{i_1}(x_i), i \in G \), is greater than \( u_{i_1} \).

**Step 3.** Obtain the set of efficient solutions \( H_1(Y_1) \) by solving the following problem over the remaining values of \( x_1 \) for any given vector \( Y_1 \):

\[
\max \{ \text{R}_1(x_1) \},
\]

s.t. \( x_1 \leq \min_{i \in G} (k_1, [a_{i_1}^{-1}(Y_{i_1})]), x_1 \) integer.

Now, one is able to obtain the solution set \( H_1(Y_1) \) through the range of values of the decision variable \( x_1 \). Observe that one is in fact obtaining first the only possible and relevant changes in the return function \( R_1(x_1) \), which correspond to the discontinuities of the set \( H_1(Y_1) \) over its range of feasible values. After eliminating the infeasible values of \( x_1 \), for a given feasible vector \( Y_1 \), one can obtain the corresponding set of solutions for all feasible values of \( x_1 \).

Let us denote by \( \Phi_1 \) the set of subpolicies \( \bar{x}_1 \), obtained from the set of all feasible points, with the property that there does not exist any other feasible integer point \( x_1 \), such that

\[
R_1(x_1) \geq R_1(\bar{x}_1) \quad \text{and} \quad A_1(x_1) \leq A_1(\bar{x}_1),
\]

with at least one strict inequality in the first expression.
The points that satisfy this property will be called resource-efficient points.

**Lemma 4.2.** Let any feasible vector of values $Y_1$ be given. Then,$$X^0(Y_1) \subseteq \Phi^1.$$ 

**Proof.** Suppose that there exists a point $x^0 \in X^0(Y_1)$ such that $x^0 \notin \Phi^1$. $x^0 \notin \Phi^1$ implies that there exists a feasible point $x_1$ such that
$$R^1(x_1) \geq R^1(x^0),$$
with at least one strict inequality. This contradicts the fact that $x^0$ is an efficient solution. 

The above lemma and definition will be helpful in improving the general modified scheme given previously when considering more than a single-stage problem. The new scheme would leave steps 1 and 2 the same and replace Step 3 by the following step.

**Step (3)’.** Obtain the set $\Phi^1$, and store it for future stages. Observe that this modification does not explicitly require finding the set $X^0(Y_1)$ for any particular value(s) of $Y_1$.

The same reasoning, that led one to conclude that the solution set as a function of $Y_1, H_1(Y_1)$, behaves in a stepwise manner and to devise the procedure just described, can be used inductively to make the same conclusions for the $m$-stage problem [see Villarreal (Ref. 17)]. The main results for this $m$-stage procedure are given omitting the proofs.

**Theorem 4.1.** The functional set of $Y_m, H_m(Y_m)$, is a step functional set which varies due to corresponding changes in the value of the point $(x_1^0, \ldots, x_{m-1}^0, x_m)$ formed by all feasible values of $x_m$ and efficient points $(x_1^0, \ldots, x_{m-1}^0) \in X^0(Y_{m-1} = Y_m - A^m(x_m))$.

Let us denote by $\Phi^m$ the set of subpolicies at stage $m$ such that, if $x^0 \in \Phi^m$, there is no other $x$ satisfying
$$R^m(x_n)\Delta, \ldots, \Delta R^1(x_1) \geq R^m(x_m^0)\Delta, \ldots, \Delta R^1(x_1^0),$$
with at least one strict inequality in the first expression.

**Theorem 4.2.** Let any feasible vector of values $Y_m$ be given. Then,$$X^0(Y_m) \subseteq \Phi^m.$$
Since
\[ X^0(Y_{m-1} = Y_m - A^m(x_m)) \subseteq \Phi^{m-1}, \]
for any vector of values \( Y_{m-1} \), then
\[ (x^0_1, \ldots, x^0_{m-1}) \in \Phi^{m-1}. \]
As a result, all the feasible points that can be formed will be contained in the set of points composed of feasible values of \( x_m \) and points
\[ (x^*_1, \ldots, x^*_{m-1}) \in \Phi^{m-1}. \]

With these results, one is now able to structure a general procedure that does not require the explicit definition of vectors of values \( Y_n, n = 1, \ldots, N - 1 \), in order to obtain the corresponding set of solutions for each vector of values \( Y_N (\leq b) \). This general scheme is outlined below.

**Step 0.** Let \( m = 1 \), and let \( \Phi^0 \) be an empty set.

**Step 1.** Obtain the set of \( m \)-dimensional integer points \((x_1, \ldots, x_m)\) such that
\[ 0 \leq x_m \leq k_m \quad \text{and} \quad (x_1, \ldots, x_{m-1}) \in \Phi^{m-1}. \]

**Step 2.** Eliminate all those \( m \)-dimensional points that are infeasible, i.e. those for which any component \( i \) of their resource-consumption vector satisfies
\[ \sum_{n=1}^{m} a_{in}(x_n) \geq b_i + \sum_{n=m+1}^{N} \left| a_{in}(k_n) \right|. \]

**Step 3.** Obtain the set of resource efficient points \( \Phi^m \) via pairwise comparison.

**Step 4.** If \( m = N \), then go to Step 5. Otherwise, replace \( m \) by \( m + 1 \), and go to Step 1.

**Step 5.** For any given vector of values \( Y_N (\leq b) \), obtain the set of efficient solutions by solving the following problem via pairwise comparisons of feasible points in \( \Phi^N \):
\[ V \text{-max} \left\{ R^1(x_1) \Delta, \ldots, \Delta R^N(x_N) \mid (x_1, \ldots, x_N) \in \Phi^N \right\} \left\{ \sum_{n=1}^{N} A^n(x_n) \leq Y_N \right\}. \]

Observe that this procedure is very similar to that given by Marsten and Morin (Ref. 24) and by Morin and Marsten (Ref. 25) for the maximization of specific single-objective problems. As pointed out before, the procedure can be applied to obtain the set of Pareto-optimal solutions for
any vector of initial resources \( Y_N \) \((\leq b)\). Thus, at the final stage, one could obtain the solution to a parametric multicriteria programming problem on the right-hand side.

5. Incorporation of Bounds

The following definitions will allow the use of bounds to make the above recursions more efficient in the case of discrete programming problems.

**Definition 5.1.** A set of upper bounds to the solution of a multicriteria programming problem is a set of points that satisfy the following conditions:

(i) each element is either efficient or dominates at least one of the efficient solutions of the problem;

(ii) each efficient solution of the problem is dominated by at least one member of the set, or it is indeed a member of the set.

**Definition 5.2.** A set of lower bounds to the set of efficient solutions of a multicriteria programming problem is a set of points such that each element is either efficient or is dominated by at least one efficient solution of the problem.

Let \( LB \) denote a set of lower bounds for the solution of problem \( P \). Consider \( W \) as the set of all possible policies and subpolicies obtained by concatenating individual decisions \( x_n \). Let \( w \in W \) stand for an individual policy or subpolicy, i.e.,

\[ w = \{x_1, \ldots, x_n\}. \]

One can show that the return and transformation functions can be expressed in terms of the policy space; i.e., for any \( n \), one has that

\[ R^n(x_n, Y_n) = \tilde{R}^n(x_n, \ldots, x_N, Y_N), \]

\[ H_{n-1}(Y_{n-1}) = \tilde{H}_{n-1}(x_n, \ldots, x_N, Y_N), \]

\[ t_n(x_n, Y_n) = \tilde{t}_n(x_n, \ldots, x_N, Y_N). \]

Let \( W(Y_n) \) denote the set of efficient subpolicies such that, when applied to \( Y_0 \), one obtains the vector of states \( Y_n \), i.e.,

\[ W(Y_n) = \{(x_1, \ldots, x_n) \mid Y_n = (\tilde{t}_1(x_n, \ldots, x_1, Y_0))^{-1}\}. \]

Also, let \( X(Y_n, Y_N) \) denote the completion set of feasible subpolicies such
that, when applied to $Y_n$, a final state vector $Y_N$ is obtained, i.e.,

$$X(Y_n, Y_N) = \{(x_{n+1}, \ldots, x_N) | Y_N = (i_n(x_N, \ldots, x_n, Y_N))^{-1}\}.$$ 

One completion set of particular interest is $X(Y_n, b)$. The problem of determining the efficient set of feasible subpolicies for $X(Y_n, b)$ will be called the residual or remaining problem, to be denoted by $RS_{n+1}(b, Y_n)$. Let $UB_{n+1}(Y_n)$ denote a set of upper bounds for $RS_{n+1}(b, Y_n)$.

**Theorem 5.1.** Let an efficient subpolicy for the $n$th stage problem with vector of resources $Y_n$, say $x$, be available. Let its $p$-dimensional return function values be denoted by $H_{x,n}$. If, for every element $g_k \in H_{x,n} \oplus UB_{n+1}(Y_n)$, there exists an element $LB_j \in LB$ such that

$$g_k \leq LB_j,$$

with at least one strict inequality, then the subpolicy $x$ cannot be part of an efficient policy for $P$.

**Proof.** Any member of the set $LB$ is either efficient or is dominated by at least one associated efficient solution for the original problem $P$. Thus, any point dominated by any member of the set $LB$ is also dominated by an (associated) efficient solution. For the case in which the set of upper bounds for the residual problem contains elements such that each one dominates each and every efficient point of the residual problem, one has that

$$H_{x,n} \oplus UB_{n+1}(Y_n) \supseteq H_{x,n} \oplus RS_{n+1}(b, Y_n),$$

with at least one strict inequality. Otherwise, each solution of the residual problem is dominated by at least one member of the set of upper bounds. In either case, each member of $H_{x,n} \oplus RS_{n+1}(b, Y_n)$ is dominated by at least one member of $H_{x,n} \oplus UB_{n+1}(Y_n)$. Thus, if (1) holds, each completion of subpolicy $x$ is dominated by an efficient policy of $P$. 

Morin and Marsten (Ref. 25) develop similar results for the single objective case. Using this result will enable one to eliminate (fathom) subpolicies which will not lead to efficient policies at various stages during
the procedure. This will, in turn, decrease the computational storage requirements since these policies will not be stored.

Of course, the best set of lower bounds would be the actual set of efficient solutions. A subset of these may be obtained by solving the single-objective problem resulting from forming a convex combination of the original objectives. Also, approximate solutions to such problems solved by a heuristic approach will provide valid lower bounds at potentially great savings. By choosing different convex combinations, a set of solutions will result. Of course, the number of such solutions to generate will depend on the computational advantages attained by the use of the extra lower bounds.

A good set of upper bounds will include any provably efficient solutions of the original problem. These may be obtained as discussed above. A proper subset of efficient solutions does not suffice as a set of upper bounds, since the second condition of Definition 5.1 would not be met. One way of meeting this second condition is to obtain the maximum of each objective individually for the residual or remaining problem of interest and forming a return vector which clearly dominates all solutions of the residual problem. An upper bound on each objective calculated by solving any appropriate relaxation of the residual problem would also suffice at potentially large computational savings.

6. Computational Results

To illustrate the applicability of the previous results, the imbedded state approach was programmed in FORTRAN IV and used to solve a sample of 0-1 multicriteria multidimensional linear knapsack problems. This type of problem is defined as follows:

\[
\text{V-max}\{Cx\},
\]

s.t. \[Ax \leq b, x_n = 0, 1, n = 1, \ldots, N,\]

where the matrices \(C\) and \(A\) are \(p \times N\) and \(M \times N\) matrices of nonnegative elements. The objective and constraint matrix coefficients were generated from a uniform distribution with range \([0, 99]\). The density of the constraint matrix was 90%, and the right-hand side vector of resources was chosen to be equal to 0.25, 0.50, and 0.75 times the sum of the corresponding row coefficients. The procedure was programmed in FORTRAN IV and run on a CDC Cyber 174 computer.

The set(s) of upper bounds are obtained by setting the variables of the residual problem to one. In this case, these sets consist of only one
element which will be common for all the partial policies at stage $n$. The set(s) of lower bounds is (are) obtained by computing feasible solutions for the problem

$$\max \{ \lambda C x \} ,$$

s.t. $Ax \leq b, x_n = 0, 1, n = 1, \ldots, N,$

where

$$\lambda \in \Lambda = \left\{ \lambda \mid \lambda_i > 0, i = 1, \ldots, p, \sum_{i=1}^{p} \lambda_i = 1 \right\} .$$

This set may consist of any number of members. The feasible solutions are determined by using the heuristic approach of Loulou and Michaelides (Ref. 27).

In using the bounding results suggested, it was decided to employ them at every stage of the procedure after Stage 3. The set of lower bounds is computed as an initial step, while the set(s) of upper bounds are calculated at each stage and the conditions of Theorem 5.1 are tested.

Table 1 shows the means and ranges of the solution times and the number of efficient solutions for bicriterion 0–1 integer programming problems of various sizes without the application of bounds. A sample of five problems was selected for each combination of number of constraints $M$, variables $N$, and specific $b$ values. Note that the algorithm performs satisfactorily for problems with $b$ value equal to 0.25 times the sum of the coefficients of the associated row; it becomes practically inadequate as the $b$ value increases. For example, for problems with $M = 10$ and $N = 10$, one has that the means of the solution times are 0.194, 9.152, and 115.710 CPU seconds for values of $b$ equal to 0.25, 0.50, and 0.75 times the sum of the coefficients of the associated rows. This significant increase in solution time is mainly due to the fact that, as the $b$ value increases, the number of feasible points also increases and the size of the set of resource-efficient points also becomes larger. Hence, the number of points to be tested for dominance increases rapidly, and thus the overall solution time increases rapidly. Table 2 shows information on a sample of 12 problems with different $b$ values and illustrates how the resource-efficient set increases at each stage for each of the different $b$ values.

Notice that the residual problem depends on the state vector $Y_n$. However, using this rule to compute upper-bound vectors results in having only one common element for all the residual problems at stage $n$. This is given by

$$\sum_{j=n+1}^{N} C_i^j x_{p_j} \quad \text{where} \quad C_i^j = (c_{1p}, \ldots, c_{p_j})^j .$$
Table 1. Solution times and efficient set sizes for 0-1 bicriterion problems.

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>b</th>
<th>X</th>
<th>R</th>
<th>Ψ</th>
<th>Time</th>
<th>EF (P)</th>
<th>X</th>
<th>R</th>
<th>EF (P)</th>
<th>X</th>
<th>R</th>
<th>EF (P)</th>
<th>X</th>
<th>R</th>
<th>EF (P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>0.25</td>
<td>1.2</td>
<td>0.058</td>
<td>0.05-0.06</td>
<td>2.4</td>
<td>3-5</td>
<td>0.078</td>
<td>0.07-0.08</td>
<td>1.6</td>
<td>1-2</td>
<td>0.120</td>
<td>0.11-0.13</td>
<td>3.2</td>
<td>2-5</td>
<td>0.256</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>0.50</td>
<td>2.0</td>
<td>0.066</td>
<td>0.06-0.07</td>
<td>3.4</td>
<td>2-5</td>
<td>0.140</td>
<td>0.11-0.16</td>
<td>2.6</td>
<td>2-3</td>
<td>0.278</td>
<td>0.16-0.88</td>
<td>4.4</td>
<td>2-8</td>
<td>0.280</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>0.75</td>
<td>1.4</td>
<td>0.075</td>
<td>0.07-0.08</td>
<td>3.2</td>
<td>2-4</td>
<td>0.360</td>
<td>0.26-0.47</td>
<td>2.8</td>
<td>2-4</td>
<td>0.056</td>
<td>0.28-4.92</td>
<td>4.0</td>
<td>3-6</td>
<td>21.650</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>0.25</td>
<td>1.0</td>
<td>0.060</td>
<td>0.05-0.06</td>
<td>1.8</td>
<td>1-2</td>
<td>0.082</td>
<td>0.06-0.08</td>
<td>2.4</td>
<td>1-4</td>
<td>0.135</td>
<td>0.13-0.14</td>
<td>2.2</td>
<td>2-3</td>
<td>0.936</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>0.50</td>
<td>1.8</td>
<td>0.084</td>
<td>0.08-0.10</td>
<td>2.4</td>
<td>2-3</td>
<td>0.392</td>
<td>0.32-0.51</td>
<td>3.0</td>
<td>1-5</td>
<td>3.982</td>
<td>2.74-6.03</td>
<td>3.8</td>
<td>2-6</td>
<td>42.980</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>0.75</td>
<td>1.8</td>
<td>0.084</td>
<td>0.08-0.10</td>
<td>2.4</td>
<td>2-3</td>
<td>0.392</td>
<td>0.32-0.51</td>
<td>3.0</td>
<td>1-5</td>
<td>3.982</td>
<td>2.74-6.03</td>
<td>3.8</td>
<td>2-6</td>
<td>42.980</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>0.25</td>
<td>1.0</td>
<td>0.060</td>
<td>0.05-0.06</td>
<td>1.8</td>
<td>1-2</td>
<td>0.082</td>
<td>0.06-0.08</td>
<td>2.4</td>
<td>1-4</td>
<td>0.135</td>
<td>0.13-0.14</td>
<td>2.2</td>
<td>2-3</td>
<td>0.936</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>0.50</td>
<td>1.8</td>
<td>0.084</td>
<td>0.08-0.10</td>
<td>2.4</td>
<td>2-3</td>
<td>0.392</td>
<td>0.32-0.51</td>
<td>3.0</td>
<td>1-5</td>
<td>3.982</td>
<td>2.74-6.03</td>
<td>3.8</td>
<td>2-6</td>
<td>42.980</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>0.75</td>
<td>1.8</td>
<td>0.084</td>
<td>0.08-0.10</td>
<td>2.4</td>
<td>2-3</td>
<td>0.392</td>
<td>0.32-0.51</td>
<td>3.0</td>
<td>1-5</td>
<td>3.982</td>
<td>2.74-6.03</td>
<td>3.8</td>
<td>2-6</td>
<td>42.980</td>
</tr>
</tbody>
</table>

EF (P) = number of efficient points; X = mean; R = range.
Table 2. Stagewise growth of the number of resource-efficient points.

<table>
<thead>
<tr>
<th>No.</th>
<th>$M$</th>
<th>$N$</th>
<th>$p$</th>
<th>$b$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>8</td>
<td>2</td>
<td>0.25</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>8</td>
<td>2</td>
<td>0.25</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>8</td>
<td>2</td>
<td>0.25</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>10</td>
<td>2</td>
<td>0.25</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td>14</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>10</td>
<td>2</td>
<td>0.50</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>20</td>
<td>37</td>
<td>60</td>
<td>95</td>
<td>133</td>
<td>190</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>10</td>
<td>2</td>
<td>0.75</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>60</td>
<td>109</td>
<td>207</td>
<td>362</td>
<td>615</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>10</td>
<td>2</td>
<td>0.75</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>118</td>
<td>209</td>
<td>373</td>
<td>598</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>10</td>
<td>3</td>
<td>0.50</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>60</td>
<td>104</td>
<td>208</td>
<td>339</td>
<td>537</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>10</td>
<td>3</td>
<td>0.50</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>60</td>
<td>108</td>
<td>207</td>
<td>267</td>
<td>385</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>10</td>
<td>3</td>
<td>0.50</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>30</td>
<td>60</td>
<td>113</td>
<td>175</td>
<td>243</td>
<td>382</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td>10</td>
<td>3</td>
<td>0.75</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>125</td>
<td>235</td>
<td>451</td>
<td>718</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>10</td>
<td>3</td>
<td>0.75</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>28</td>
<td>56</td>
<td>108</td>
<td>194</td>
<td>366</td>
<td>637</td>
</tr>
</tbody>
</table>

$p =$ number of objective functions.
Another interesting aspect of the computational characteristics of the algorithm is how it behaves as the number of objective functions, constraints, and variables change. Table 3 shows some evidence that supports the idea that, as the number of objective functions increases so does the solution time. Table 1 gives some evidence that when (either one or both) the number of constraints and variables increase, the solution times also increase. Further, one can notice that the increase in solution times due to the increase in the number of variables is exponential. The increase in solution time due to increases of the number of constraints and objective functions is mainly due to the fact that the feasibility and dominance tests take longer to identify infeasible or dominated points, because more comparisons must be done for higher-dimensional solutions. Increasing the number of variables results in larger sets of resource-efficient points in the extra stages that must be tested for both feasibility and dominance.

Table 4 gives an idea of the performance of the procedure without bounds compared with the only one given in the literature, suggested by Bitran (Ref. 4). Even though the comparison is not uniform (because the problems, the programming language, and the computer system are not the same), it does give a general insight into the types of problems in which the methods tend to perform better. The general structure of the 0-1 problems \((p=3; M=4; N=10, 14; b = 0.25, 0.50, 0.75; A, b, c \geq 0; \text{density}=90\%)\) is the same. The dynamic recursive procedure seems to perform better for problems with \(b\) value equal to 0.25 times the sum of the coefficients of the associated rows. For problems with larger \(b\) value, Bitran's procedures perform better.

Tables 5 and 6 show the solution times of samples of ten and five problems respectively, solved by the original and the modified (bounds incorporated) recursions. In both samples, the problems have ten constraints, ten variables, two objectives functions, and \(b\) values of 0.75 and 0.50 times the sum of the associated row coefficients, respectively. Table 5 shows the solution times of the problems for which the modified dynamic programming recursion employs sets of lower bounds \(LB\) consisting of two, three, and four elements. The solution times using the original dynamic programming recursion corresponds to those shown in the column with heading "no bounds". Notice that the increase in the size of the set of lower bounds (from two to four) is not of much help for fathoming purposes (with various exceptions). This increase in the number of members of the set of lower bounds is accomplished by considering more \(\lambda\) values with the characteristics previously mentioned. The increase in the solution times for the cases in which the set of lower bounds becomes larger is due to the increase in the computational times for the extra members, and their lack of effectiveness for fathoming purposes (eliminating subpolicies).
Table 3. Effects of varying number of objective functions.

<table>
<thead>
<tr>
<th>$p$</th>
<th>Problem number</th>
<th>EF ($P$) No.</th>
<th>$\tilde{X}$</th>
<th>Sol. time</th>
<th>$\tilde{X}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>16</td>
<td></td>
<td>11.99</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
<td>10.6</td>
<td>4.41</td>
<td>8.85</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>12</td>
<td></td>
<td>8.97</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>6</td>
<td></td>
<td>12.28</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>12</td>
<td>9.0</td>
<td>7.34</td>
<td>9.70</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>9</td>
<td></td>
<td>9.49</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>6</td>
<td></td>
<td>14.72</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>19</td>
<td>11.3</td>
<td>15.32</td>
<td>12.50</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>9</td>
<td></td>
<td>7.48</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>20</td>
<td></td>
<td>6.88</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>23</td>
<td>20.6</td>
<td>16.34</td>
<td>12.83</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>19</td>
<td></td>
<td>15.27</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>13</td>
<td>19</td>
<td></td>
<td>15.59</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>28</td>
<td>21.0</td>
<td>21.12</td>
<td>17.50</td>
</tr>
<tr>
<td>7</td>
<td>15</td>
<td>16</td>
<td></td>
<td>15.81</td>
<td></td>
</tr>
</tbody>
</table>

$M = 4$, $N = 10$, $b = 0.50$, density = 90%.

The number of efficient solutions of these problems varied from four to eight. Table 6 gives the solution times for five problems using the dynamic recursions with and without the bounding results. In this case, the set LB consists of only two elements. The size of the efficient set of these problems varies from two to four. Both tables show that the use of bounds results in significant improvement in solution times over the original dynamic programming recursive equations.

A detailed illustration of the reason why the sample of problems shows this improvement in solution times is given in Table 7. This table illustrates the sizes of the sets of partial and fathomed policies at every stage. Notice the difference in the sizes of these sets under both recursions. These differences result in direct savings in terms of computer storage. The problems shown are a sample of four from the ten problems of Table 5.
Table 4. Comparison with Bitran’s algorithm.

\[
\begin{array}{cccccccc}
\hline
b & \tilde{X} & R & \tilde{X} & R & \tilde{X} & R & \tilde{X} & R \\
\hline
0.25 & 1.40 & 1.11-1.98 & 0.29 & 0.23-0.35 & 28.10 & 27.87-28.29 & 5.86 & 5.69-6.15 \\
0.50 & 1.64 & 1.52-1.98 & 0.38 & 0.41-12.80 & 19.61 & 17.63-20.41 & 23.40 & 22.41-24.06 \\
0.75 & 0.77 & 0.63-0.81 & 78.50 & 50.42-105.78 & 4.79 & 3.33-6.85 & 8.82 & 7.46-10.91 \\
\hline
M = 4, \ p = 3, \ \text{density} = 90\%. \\
* \text{Combined results: problems with } b = 0.25 \text{ solved using Bitran 2 and the rest with Bitran 1.} \\
† \text{Bitran 1 using simplex.}
\end{array}
\]
Table 5. Empirical results of varying the number of lower-bounds, $b = 0.75$.

<table>
<thead>
<tr>
<th>Problem number</th>
<th>No bounds</th>
<th>Two bounds</th>
<th>Three bounds</th>
<th>Four bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sol. Time</td>
<td>Sol. time</td>
<td>Sol. time</td>
<td>Sol. Time</td>
</tr>
<tr>
<td>1</td>
<td>128.17</td>
<td>9.66</td>
<td>8.08</td>
<td>10.01</td>
</tr>
<tr>
<td>2</td>
<td>106.82</td>
<td>8.41</td>
<td>8.80</td>
<td>12.11</td>
</tr>
<tr>
<td>3</td>
<td>116.75</td>
<td>2.71</td>
<td>3.02</td>
<td>3.05</td>
</tr>
<tr>
<td>4</td>
<td>93.71</td>
<td>6.42</td>
<td>4.80</td>
<td>5.83</td>
</tr>
<tr>
<td>5</td>
<td>135.02</td>
<td>17.57</td>
<td>9.47</td>
<td>9.38</td>
</tr>
<tr>
<td>6</td>
<td>126.85</td>
<td>5.29</td>
<td>5.49</td>
<td>5.19</td>
</tr>
<tr>
<td>7</td>
<td>117.16</td>
<td>8.33</td>
<td>8.73</td>
<td>6.45</td>
</tr>
<tr>
<td>8</td>
<td>110.63</td>
<td>4.87</td>
<td>5.20</td>
<td>4.83</td>
</tr>
<tr>
<td>9</td>
<td>90.51</td>
<td>6.65</td>
<td>4.98</td>
<td>6.45</td>
</tr>
<tr>
<td>10</td>
<td>51.83</td>
<td>1.11</td>
<td>1.36</td>
<td>1.51</td>
</tr>
</tbody>
</table>

$M = 10, N = 10, p = 2.$

Table 6. Empirical results of varying the number of lower bounds, $b = 0.50$.

<table>
<thead>
<tr>
<th>Problem number</th>
<th>No bounds</th>
<th>Two bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sol. Time</td>
<td>Sol. time</td>
</tr>
<tr>
<td>1</td>
<td>8.68</td>
<td>3.59</td>
</tr>
<tr>
<td>2</td>
<td>7.35</td>
<td>4.80</td>
</tr>
<tr>
<td>3</td>
<td>7.93</td>
<td>1.25</td>
</tr>
<tr>
<td>4</td>
<td>9.00</td>
<td>2.62</td>
</tr>
<tr>
<td>5</td>
<td>7.80</td>
<td>1.14</td>
</tr>
</tbody>
</table>

$M = 10, N = 10, p = 2.$
Table 7. Illustration of the behavior of the sets of partial and fathomed points.

<table>
<thead>
<tr>
<th>Problem number</th>
<th>Method</th>
<th>Stage number</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>M</td>
<td>F</td>
<td>M</td>
<td>F</td>
<td>M</td>
<td>F</td>
<td>M</td>
<td>F</td>
<td>M</td>
</tr>
<tr>
<td>1</td>
<td>DPA</td>
<td>8</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>31</td>
<td>0</td>
<td>61</td>
<td>0</td>
<td>115</td>
</tr>
<tr>
<td>1</td>
<td>BDPA</td>
<td>8</td>
<td>0</td>
<td>15</td>
<td>1</td>
<td>26</td>
<td>3</td>
<td>46</td>
<td>5</td>
<td>70</td>
</tr>
<tr>
<td>3</td>
<td>DPA</td>
<td>8</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>32</td>
<td>0</td>
<td>58</td>
<td>0</td>
<td>112</td>
</tr>
<tr>
<td>3</td>
<td>BDPA</td>
<td>8</td>
<td>0</td>
<td>14</td>
<td>2</td>
<td>22</td>
<td>6</td>
<td>28</td>
<td>11</td>
<td>31</td>
</tr>
<tr>
<td>5</td>
<td>DPA</td>
<td>8</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>32</td>
<td>0</td>
<td>63</td>
<td>0</td>
<td>116</td>
</tr>
<tr>
<td>5</td>
<td>BDPA</td>
<td>8</td>
<td>0</td>
<td>15</td>
<td>1</td>
<td>26</td>
<td>4</td>
<td>50</td>
<td>1</td>
<td>75</td>
</tr>
<tr>
<td>10</td>
<td>DPA</td>
<td>8</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>32</td>
<td>0</td>
<td>56</td>
<td>0</td>
<td>96</td>
</tr>
<tr>
<td>10</td>
<td>BDPA</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>10</td>
<td>6</td>
<td>15</td>
<td>4</td>
<td>11</td>
</tr>
</tbody>
</table>

DPA = dynamic programming approach; BDPA = dynamic programming approach with bounds; M = partial policies; F = fathomed policies (eliminated subpolicies).
References


