An Algorithm for the Discrete Bilevel Programming Problem

Jonathan F. Bard
Operations Research Group, Department of Mechanical Engineering, University of Texas, Austin, Texas 78712

James T. Moore
Air Force Institute of Technology, Wright-Patterson AFB, Dayton, Ohio 45433

The bilevel programming problem (BLPP) is an example of a two-stage, noncooperative game in which the first player can influence but not control the actions of the second. This article addresses the linear formulation and presents a new algorithm for solving the zero-one case. We begin by converting the leader's objective function into a parameterized constraint, and then attempt to solve the resultant problem. This produces a candidate solution that is used to find a point in the BLPP feasible reagion. Incremental improvements are sought, which ultimately lead to a global optimum. An example is presented to highlight the computations and to demonstrate some basic characteristics of the solution. Computational experience indicates that the algorithm is capable of solving problems with up to 50 variables in a reasonable amount of time.

1. INTRODUCTION

The bilevel programming problem (BLPP) is a model for a leader-follower game in which two players try to maximize their individual objective functions, $F(x, y)$ and $f(x, y)$, respectively [4, 14]. Play is defined as sequential and the mood as noncooperative. The decision variables are partitioned amongst the players in such a way that neither can dominate the other. The leader goes first and through his choice of $x \in X \subseteq R^{n_1}$, is able to influence but not control the actions of the follower. This is achieved by reducing the set of feasible choices available to his opponent. Subsequently, the follower reacts to the leader's decision by choosing a $y \in Y \subseteq R^{n_2}$ in an effort to maximize his payoff. In so doing, he indirectly affects the leader's outcome.

In this article we consider the integer linear version of this (Stackelberg) game with binary restrictions on the variables. The problem is

$$\max_x F(x, y) = c^1x + c^2y, \tag{1a}$$

Naval Research Logistics, Vol. 39, pp. 419–435 (1992)
 CCC 0894-069X/92/030419-17$04.00

where y solves

$$\underset{y}{\text{Max}}\ f(x, y) = d^1x + d^2y, \tag{1b}$$

$$\text{subject to } g(x, y) = Ax + By \leq b, \tag{1c}$$

$$x \in X = \{x_j \text{ binary: } j = 1, \ldots, n_1\},$$

$$y \in Y = \{y_j \text{ binary: } j = 1, \ldots, n_2\}, \tag{1d}$$

where c^1 and d^1 are $(1 \times n_1)$, c^2 and d^2 are $(1 \times n_2)$, A is $(m \times n_1)$, B is $(m \times n_2)$, and b is $(m \times 1)$. The elements of these arrays are assumed to be integer constants. It should be noted that the term d^1x in the follower's objective function can be omitted without affecting the solution of the BLPP. This follows because once x is chosen, the resulting problem [(1b)–(1d)] becomes an zero-one linear program in y only. [This observation explains the difference between (1b) and (4a) in Section 3.]

The BLPP can also be interpreted as a standard optimization problem with implicitly defined constraints. This view implies that the follower must solve a mathematical program parameterized in the leader's decision variables. In either case, most of the ongoing research in this area centers on the development of algorithms for solving the continuous problem where F and g are linear, and f is either linear or quadratic [2, 5, 8, 9]. A series of heuristics for the mixed-integer linear BLPP is presented in [12]. Our efforts complement this work by providing a methodology for solving the zero-one BLPP. Implicit enumeration provides the foundation.

In fact, the proposed algorithm can be easily modified to solve (1) for the more general case where the y's are unrestricted integer variables. However, our present discussion and implementation are limited to the binary case.

In the next section, a production planning problem is presented to demonstrate the potential use of bilevel programming. In Section 3, notation is given and the leader-follower game is reformulated as a right-hand-side parametric program. The latter forms the basis of the algorithm presented in Section 4. An example is worked out in Section 5 where some of the peculiarities of the solution are demonstrated. Section 6 contains our experimental results and a discussion of the algorithm's performance.

2. AN APPLICATION

Most applications of bilevel programming that have appeared in the literature deal with central economic planning at the regional or national level. In this context, the government is considered the leader and controls a set of policy variables such as tax rates, subsidies, import quotas, and price supports. The particular industry targeted for regulation is viewed as the follower. In most cases, the follower tries to maximize net income subject to the prevailing technological, economic, and governmental constraints. Possible leader objectives include maximizing employment, maximizing production of a given product, or minimizing the use of certain resources.

The early work of Candler and Norton [7], focusing on agricultural development in northern Mexico, illustrates how bilevel programming can be used to analyze the dynamics of a regulated economy. Similarly, Fortuny-Amat and McCarl [9] present a regional model that pits fertilizer suppliers against local farmer communities, while Aiyoshi and Shimizu [1] discuss resource allocation in a decentralized firm. In the case of the latter, a central unit supplies resources to its manufacturing facilities, which make decisions concerning production mix and output. Organizational procedures and conflicting objectives over efficiency, quality, and performance lead to a hierarchical formulation.

To further illustrate the use of bilevel programming, consider the situation where a manufacturer of n products wishes to determine a production mix, $x \in R^{n \times T}$, that maximizes his profits over a finite horizon T, in the face of m resource constraints. Assume that the demand at time t, $d_t \in R^n$, is "inexact" (Soyster [15]), known only to lie in the set D_t whose boundaries are subject to the influences of advertising. For example, D_t might specify upper and lower bounds on the demand for each product in period t. Now suppose the existence of a sole customer whose demand (requirements) for the n products is a function of the manufacturer's expenditure on advertising, $v \in R^{n \times T}$, where "advertising" is intended to include all activities, such as marketing, the use of extended warranties, and temporary on-site troubleshooting, that might sway the customer's purchase decisions. (A motivating example concerns the manufacture of specialized subsystems for an auto maker by a supplier new to the market. It is not unusual for a start-up firm producing specialized parts to have a single customer in its first few years of operation.) Further assume that the customer is rational and wishes to meet his demand at minimum cost. If subcontracting to meet shortages is permitted, and market conditions require the manufacturer to commit first for the entire T-period horizon, the following two-level problem results:

$$\max_{x,v} \sum_{j=1}^{n} \sum_{t=1}^{T} p_{jt} d_{jt} - \sum_{j=1}^{n} \sum_{t=1}^{T} [c_{jt} x_{jt} + h_{jt} I_{jt} + s_{jt} S_{jt} + v_{jt}] \tag{2a}$$

$$\text{subject to } \sum_{j=1}^{n} a_{ijt} x_{jt} \leq b_{it}, \qquad \forall\, i, t \tag{2b}$$

$$x_{jt} \geq 0, \qquad v_j = (v_{j1}, \ldots, v_{jT}) \in V_j, \qquad \forall\, j, t, \tag{2c}$$

where d, I, and S solve

$$\max_{d,I,S} \sum_{j=1}^{n} \sum_{t=1}^{T} -p_{jt} d_{jt}, \tag{2d}$$

$$\text{subject to } d_{jt} - I_{j,t-1} + I_{jt} - S_{jt} = x_{jt}, \qquad \forall\, j, t \tag{2e}$$

$$S_{jt} \geq 0, \qquad I_{jt} \geq 0, \qquad I_{j0} = 0, \qquad \forall\, j, t \tag{2f}$$

$$d_t = (d_{1t}, \ldots, d_{nt}) \in D_t(v), \qquad \forall\, t, \tag{2g}$$

where

x_{jt} is the amount of product j manufactured in period t
a_{ijt} is the (known) amount of resource i required to make a unit of product j in period t
b_{it} is the (known) amount of resource i available in period t
I_{jt} is the amount of inventory of product j at the end of period t; it is assumed that the initial inventory I_{j0} is 0
s_{jt} is the amount of shortage of product j in period t
v_{jt} is the amount spent on advertising product j in period t
V_j is the set constraining advertising expenditures for product j
p_{jt} is the (known) selling price of product j in period t
c_{jt} is the (known) unit cost of manufacturing product j in period t
h_{jt} is the (known) unit holding cost of product j in period t
s_{jt} is the (known) unit cost of subcontracting product j in period t
d_{jt} is the leader's (inexact) demand forecast (alternatively, the customer's demand) for product j in period t
$D_t(v)$ is the set in which the n-dimensional vector d_t lies in period t

The objective function (2a) represents the difference between total revenues and the following costs: production, inventory, shortage, and advertising. Prices, p_{jt}, are assumed to be fixed and outside the control of the manufacturer. Constraint (2b) places certain restrictions on the use of resources, while limits on advertising are implicitly established in (2c), and affect demand through (2g). In practice, $D_t(v)$ would depend only on $\{v_{js} : \forall\, s \leq t\}$.

For x and v fixed, the customer tries to satisfy his demand at minimum cost by solving (2d)–(2g). As a direct consequence of the assumption that all demand must be met, the customer effectively controls inventory and shortages. This can be seen from the material balance equation (2e), which indicates that once the production and advertising decisions are taken, demand must be satisfied from a combination of inventory and subcontracting.

If D_t and V_j are polyhedral, and all the variables are discrete and bounded, then the usual devices [10] for passing from integer to binary variables can be used to transform problem (2) into a zero-one BLPP (1). In our model, the manufacturer assumes the role of leader and begins by selecting a production mix and advertising strategy for each point in time. In response, the customer makes a set of purchase decisions to minimize the cost of meeting his demand. This places him in the role of follower. (For an example relating to a manufacturer of electric power supplies, the interested reader is referred to Bard and Moore [3].)

3. NOTATION AND PROBLEM REFORMULATION

In the development it will be assumed that both the leader and follower have full knowledge of each other's objective function, but that cooperation is not permitted. This eliminates the use of side payments, and implies that solutions to (1) may be dominated, i.e., not Pareto-optimal [18].

Reference to the following notation will be made throughout the development.

BLPP Constraint Region:

$$\Omega = \{(x, y) : Ax + By \leq b, x \in X, y \in Y\}.$$

Follower's Feasible Region for $x \in X$ Fixed

$$\Omega(x) = \{y \in Y : Ax + By \leq b\}.$$

Follower's Rational Reaction Set

$$\Psi(x) = \{y: y \text{ solves: } \max[f(x, y) : y \in \Omega(x)]\}.$$

Inducible Region

$$\text{IR} = \{(x, y): x \in X, y \in \Psi(x)\}.$$

In order to ensure that the game is well defined, we impose two additional requirements: (1) Ω is nonempty, and (2) for each decision of the leader, the follower has a feasible response; that is, $\Omega(x) \neq 0$. The rational reaction set, $\Psi(x)$, consists of the "best" of these responses, while the inducible region, IR, represents the set over which the leader may optimize if given control of all the variables. This implies that the BLPP can be written as

$$\max(F(x, y): (x, y) \in \text{IR}). \tag{3a}$$

Nevertheless, even with these restrictions, the BLPP may not have a solution in the conventional sense. If $\Psi(x)$ is not a singleton at the optimum (call it x^*), different values of y in $\Psi(x^*)$ may produce different objective function values for the leader, only a subset of which maximizes $F(x^*, y)$. Because the rules of the game do not permit cooperation between the players, the leader has no way of inducing the follower to choose a response in that subset.

In order to deal with this situation, Basar and Olsder [4] suggest a conservative strategy that redefines the solution criterion for (1) as

$$\max_{(x,\ y)\in IR} \min_{y\in\Psi(x)} F(x, y). \tag{3b}$$

Alternatively, Bard and Falk [2] propose solving the inner minimization subproblem above with x fixed at x^* to see if $\Psi(x^*)$ is a singleton. Although we did not include this step in our algorithm, it would not be difficult to implement. Its omission in the sequel is tantamount to assuming either that $\Psi(x^*)$ is single valued or that the follower will choose from his rational reactions one that maximizes $F(x^*, y)$. The latter case represents an optimistic scenario where the "min" operator in (3b) is replaced by "max."

For the continuous version of (1), it is possible to construct an explicit representation of IR by exploiting the Kuhn-Tucker conditions associated with (1b)–(1d) for x fixed. This permits problem (3a) to be solved directly (e.g., see [1, 2, 9]). Such is not the case here. In addition, the fact that optimal solutions need

not be Pareto-optimal, militates against the use of algorithms designed for the integer multiobjective problem [6, 11, 16, 17].

The key to our algorithm is in the recognition that any solution to (1) must have y lying in the follower's rational reaction set, $\Psi(x)$. By limiting our search to this set while constantly seeking improvement in the leader's objective function we are able to quickly uncover good points in the inducible region. This is achieved by formulating and repeatedly solving instances of the following parameterized integer program (cf. [13]) derived from (1):

$$\max_{x,y} f(y) = d^2y, \tag{4a}$$

$$\text{subject to } Ax + By \leq b, \tag{4b}$$

$$F(x, y) = c^1x + c^2y \geq \alpha, \tag{4c}$$

$$\sum_{j=1}^{n_1} x_j \geq \beta, \tag{4d}$$

$$x \in X, \qquad y \in Y, \tag{4e}$$

where α and β are right-hand-side parameters initially taken as $-\infty$ and 0, respectively. Constraint (4c) forces a tradeoff between the two objective functions, and is identical to the cut used by Bialas and Karwan [5] in their complementary pivot heuristic for the continuous BLPP. (In contrast to Bialas and Karwan who require $c^2 \geq 0$, we place no restrictions on any of the problem coefficients.) Last, constraint (4d) restricts the sum of the variables controlled by the leader and although not strictly necessary, has proven to be an effective way of selecting branching variables in our implicit enumeration scheme. This is further discussed in the next section.

4. ALGORITHM

A depth-first implicit enumeration scheme centering on the leader's decision variables, and applied to problem (4), is used to solve (1). It is assumed that the reader is familiar with the concepts associated with a binary search tree. The basic idea is to successively examine points that satisfy the constraints of (4), fix the x variables at their corresponding values, and then re-solve (4) to obtain a new point in the inducible region. By adjusting α and β at each iteration, the algorithm steadily increass the leader's objective function until the problem becomes infeasible.

In order to provide a structure for enumerating the x variables, let $W = \{1, \ldots, n_1\}$, and define at the kth iteration of the algorithm a path vector P_k of length $l = |W_k|$ ($W_k \subseteq W$) corresponding to an assignment of either $x_j = 0$ or $x_j = 1$ for $j \in W_k$. The vector P_k identifies a partial solution for the variables controlled by the leader at the lth level of the search tree. Here $l = |W_k|$ is the length of the path from the root of the tree to the current node. The index set of assigned variables is denoted by W_k. The path vector indicates the order in

which the variables in W_k have been assigned, as well as their binary state. For example, if $x_3 = 0$ and $x_2 = 1$ have (in that order) been fixed at iteration k, then $W_k = \{2, 3\}$, $l = 2$, and $P_k = (\underline{3}, 2)$.

Now let

$$S_k^+ = \{j: j \in W_k \quad \text{and} \quad x_j = 1\},$$

$$S_k^- = \{j: j \in W_k \quad \text{and} \quad x_j = 0\},$$

$$S_k^0 = \{j: j \in W_k\}.$$

A completion of W_k is an assignment of binary values to the free variables controlled by the leader. The latter constitute the index set S_k^0. The algorithm never explicitly places restrictions on the follower's variables.

After k iterations, let IR^k be the set of points in the inducible region so far uncovered, and denote by $\underline{F}$ the incumbent lower bound associated with the leader's objective function. That is, $\underline{F} = \max[F(x, y): (x, y) \in \mathrm{IR}^k]$. At the start of the algorithm, all variables are free and $\underline{F} = -\infty$.

Step 0: (Initialization.) Put $k = 0$, $S_k^+ = \emptyset$, $S_k^- = \emptyset$, $S_k^0 = \{1, \ldots, n_1\}$, $\alpha = -\infty$, $\beta = 0$, and $\underline{F} = -\infty$. This creates the root of the search tree.

Step 1: (General Iteration.) Set $x_j = 1$ for $j \in S_k^+$ and $x_j = 0$ for $j \in S_k^-$. For the current values of α and β, attempt to find a feasible solution to (4). If successful, increment the counter k by 1, label the solution (x^k, y^k) and go to Step 2; otherwise fathom the current node and go to Step 6.

Step 2: (Bounding.) Fix x at x^k, relax (4c) and (4d), and solve (4a), (4b), and (4e) to get a point $(x^k, \hat{y}^k) \in IR$. Compute $F(x^k, \hat{y}^k)$ and put $\underline{F} = \max[\underline{F}, F(x^k, \hat{y}^k)]$.

Step 3: Let $J = \{j \in S_{k-1}^0: x_j^k = 1\}$. If $J = \emptyset$, set $S_k^+ = S_{k-1}^+$, $S_k^- = S_{k-1}^-$, $S_k^0 = S_{k-1}^0$, $P_k = P_{k-1}$, and go to Step 5; otherwise go to Step 4.

Step 4: (Branching.) Create $|J|$ new live nodes as follows: Append $j \in J$ to P_{k-1} in ascending order, one by one, to obtain the new live nodes and the final path P_k; set $S_k^+ = S_{k-1}^+ \cup J$, $S_k^0 = S_{k-1}^0 \backslash J$, and $S_k^- = S_{k-1}^-$. The new path P_k extends the old path by length $|J|$, and leads from the root to the new current node.

Step 5: Let $\alpha = \underline{F} + 1$. Let $\beta = 1 + |S_k^+|$. Go to Step 1.

Step 6: (Backtracking.) If no live nodes exist, go to Step 7. Otherwise backtrack from the current node (this is the most recently created live node; call the corresponding variable j'), branch on its complement by setting $x_{j'}^k = 0$, and update S_k^+, S_k^-, S_k^0, and P_k as discussed below. Put $\beta = 0$ and go to Step 1.

Step 7: (Termination.) If $\underline{F} = -\infty$, there is no feasible solution to (1). Otherwise, declare the feasible point associated with $\underline{F}$ an optimal solution.

Step 1 is designed to find a new point which is potentially bilevel feasible. The counter k is only incremented if the search is successful. The results of our testing indicate that solving (4) to optimality at this step offers little advantage, primarily because the solution [call it (x^k, y^k)] is not necessarily in the inducible region. That is, another point might exist that satisfies (4b), (4d), and (4e), but violates the cut (4c) and provides the follower with a larger objective function value than $f(y^k)$.

At Step 2, x is fixed at x^k, constraints (4c) and 4(d) are relaxed, and the resultant subproblem is solved to get a point in the inducible region. If an improvement is realized, the incumbent is replaced. The set J in Step 3 identifies

the branching variables. If J is empty, control passes to Step 5 where β is set to one plus the number of elements in S_k^+. The algorithm then returns to Step 1. Any feasible solution found at this point will be accompanied by a $J \neq \emptyset$.

Branching occurs at Step 4 where S_k^+, S_k^0, and P_k are updated. In the process, all free x variables equal to one in the solution at Step 1 are set to one, and a corresponding number of new nodes is created in the search tree. Contrary to the usual depth-first procedure, where one new node is created at each iteration, we found it advantageous to extend the tree by several levels at a time.

At Step 5, α is set to the current best known value, $\underline{F}$, of (1) and incremented by one. This guarantees that when the algorithm returns to Step 1, if a feasible solution of (4) is found, say (x^k, y^k), it will satisfy $F(x^k, y^k) > \underline{F}$. Also at Step 5, β is set to the sum of the variables controlled by the leader and is incremented by one. This ensures that at least one element of the set S_k^0 at iteration k has a value of one, thus providing a branching variable at Step 4 in the next passage through Step 3. If constraint (4d) were not included, the same x values might result when (4) is next solved at Step 1. Should this occur, the algorithm is said to have stalled because no further progress is possible without introducing additional branching rules.

In our original design, (4d) did not appear, so it was necessary to develop a procedure for selecting a branching variable. After considerable testing, the rule that showed the most promise involved finding max $\{c_j: j \in S_{k-1}^0 \text{ and } x_j^k \neq 1\}$ to get the corresponding index j. However, neither this rule nor any of the others investigated worked as well as the one actually implemented.

If problem (4) is infeasible at Step 1, the algorithm proceeds to Step 6 where the backtracking operation is performed. Note that a live node is one associated with a subproblem defined by a combination of elements in S_k^+ and S_k^- that has yet to be fully explored or fathomed at Step 1 due to infeasibility. To facilitate bookkeeping, the path P_k in the search tree is represented by an l-dimensional vector, where l is the current depth of the tree. The order of the components is determined by their "level." Indices only appear in the vector P_k if they are in S_k^+ or S_k^-, and appear underlined if they are in S_k^-. In the branching operation at Step 4, if more than one variable is selected, they are appended to the path vector in ascending order. Note that each P_k defines a unique node in the tree. Because the algorithm always branches to the left first ($x_j = 1$), backtracking is accomplished by finding the rightmost nonunderlined element of P_k, underlining it, and erasing all entries to its right. Though not explicitly stated in Step 6, a new node is added to the search tree each time the backtracking operation is performed. The newly underline entry is deleted from S_k^+ and added to S_k^-; the erased entries are deleted from S_k^- and added to S_k^0. This leads to the following result.

PROPOSITION 1: Let X^* be the set of optimal solutions for the leader, assuming an optimum exists. If $\Psi(x)$ is single valued for all $x \in X^*$, or if $f(y)$ is unique for $(x, y) \in \{x \in X^*, y \in \Psi(x)\}$, then the algorithm terminates with an optimal solution to the BLPP (1). Alternatively, a sufficient condition for the algorithm to terminate with an optimal solution is that the follower always selects a $y \in \Psi(x)$ that maximizes $F(x, y)$ for x fixed.

PROOF: The algorithm implicitly looks at all combinations of x at Steps 4 and 6; the subproblem solved at Step 2 guarantees that all incumbent solutions are in the inducible region. ■

REMARK: The conditions of Proposition 1 are not readily verifiable, so unless the subproblem suggested by Bard and Falk is solved, it cannot be determined whether the algorithm terminates with the optimum. However, by using ideas from goal programming and including the term $-\varepsilon F(x, y)$ (where $\varepsilon > 0$ is an arbitrarily small constant) in the objective function (4a), the algorithm will yield a solution to the more general problem (3b) thus obviating the need to check for multiple optima. To see this, replace $f(y)$ with $f(y) - \varepsilon F(x, y)$ in (4a) and let $\Psi'(x)$ be the set of solutions associated with problem (4a), (4b), and (4e) for x fixed. Because ε is arbitrarily small and y is discrete, it follows that $\Psi'(x) \subseteq \Psi(x)$. Hence, $(x, y') \in \text{IR}$, where $y' \in \Psi'(x)$. Thus, each time (4a), (4b), and (4e) are solved with x fixed at Step 2 to get a point in IR, the solution, call it y', maximizes $f(y)$ while concurrently *minimizing* $F(x, y)$ over all points in $\Psi(x)$.

Alternative Algorithm

Each time a feasible point (x^k, y^k) is uncovered at Step 1, a subproblem is solved at Step 2 to obtain a point $(x^k, \hat{y}^k)$ in the inducible region. Because that additional effort is wasted whenever $y^k = \hat{y}^k$, it would be helpful to know when this condition is met. If (4) is solved to optimality, the following result can be used to provide a partial check.

PROPOSITION 2: Let $(\bar{x}, \bar{y})$ solve problem (4). Then $(\bar{x}, \bar{y}) \in \text{IR}$ if

$$c^1\bar{x} + \sum_{j=1}^{n_2} (c_j^2 - \max[c_j^2, 0]) \geq \alpha. \tag{5}$$

PROOF: In order for $(\bar{x}, \bar{y})$ not to be in the inducible region, $\bar{y} \in Y$ must fail to maximize $f(y)$ subject to $By \leq b - A\bar{x}$. Thus there would exist a $\hat{y} \in Y$ such that $B\hat{y} \leq b - A\bar{x}$ and $f(\hat{y}) > f(\bar{y})$. Since $(\bar{x}, \bar{y})$ solves problem (4), it follows that $c^1\bar{x} + c^2\hat{y} < \alpha$. Now, by noting that the summation in (5) is equal to $\min_{y \in Y}[c^2y]$, which of course must be less than or equal to $c^2\hat{y}$, we see that it is impossible for $c^1\bar{x} + c^2\hat{y} < \alpha$. This contradiction proves the result. ■

In light of Proposition 2, an alternative computational scheme can be developed by modifying the first two steps of the algorithm in the following manner.

Step 1a: (General Iteration.) Set $x_j = 1$ for $j \in S_k^c$ and $x_j = 0$ for $j \in S_k$. For the current values of α and β attempt to find an optimal solution to (4). If successful, increment the counter k by 1, label the solution (x^k, y^k) and go to Step 2a; otherwise go to Step 6.

Step 2a: (Bounding.) Use Proposition 2 to try to verify that (x^k, y^k) is in the inducible region. If so, go to Step 2b; otherwise fix x at x^k, relax (4c) and (4d), and solve (4) to get a point $(x^k, \hat{y}^k) \in \text{IR}$.

Step 2b: Compute $F(x^k, \hat{y}^k)$ and put $\underline{F} = \max[\underline{F}, F(x^k, \hat{y}^k)]$.

This modification represents a tradeoff between the effort required at Step 1 to find a feasible solution to (4) versus the effort required at Step 1a to find an optimal solution to (4) with the prospect of not having to solve the integer program at Step 2a. That is, at iteration k with $|S_k^0|$ free x variables, the original algorithm first seeks a feasible solution to a binary program with $|S_k^0| + n_2$ variables and then (if successful) an optimal solution to a binary program with only n_2 variables. The modified procedure immediately seeks an optimal solution to a binary program with $|S_k^0| + n_2$ variables.

After some preliminary testing on problems with 20 leader variables, 20 follower variables, and 16 constraints, we found that the original algorithm was anywhere from 2 to 10 times faster then the alternative. Additional investigation showed that the only time the latter is more efficient is when n_2 is much greater than n_1. In this case, the subproblems at Step 2 are relatively burdensome so some advantage is gained if they do not have to be solved at each iteration.

Finally, it should be mentioned that there is nothing in either approach that limits the y variables to be binary, or their corresponding functions to be linear. Integrality is required, though, if the α cut (4c) is to be effective. The only constraining factor is the software used at Steps 1 and 2 to solve the respective subproblems. In addition, by substituting a branching rule for the β cut (4d), the algorithm can be further generalized to include the case where the x's are unrestricted integer variables.

5. EXAMPLE

A simple example is used to demonstrate how the algorithm works. Coincidently, it illustrates two interesting characteristics of the solution:

$$\max_{x \geq 0} F(x, y) = -x - y,$$

where y solves

$$\max_{y \geq 0} f(x, y) = 5x + y,$$

$$\begin{aligned} \text{subject to} \quad & -x - y/2 \leq -2, \\ & -x/4 + y \leq 2, \\ & x + y/2 \leq 8, \\ & x - 2y \leq 4 \end{aligned}$$

The optimal solution of this problem without regard to integrality is $(x^*, y^*) = (8/9, 20/9)$, with $F = -28/9$. A graph of the feasible region, Ω, indicates that $x < 8$ and $y < 4$. Alternatively, the third constraint of the inner problem shows that $x \leq 8$ (with $x = 8$ requiring $y = 0$, which the fourth constraint forbids), while four times the second constraint plus the third constraint gives $4.5y \leq 16$, or $y < 4$. Therefore, if x and y are now restricted to integer values,

a zero-one formulation can be obtained by letting $x = x_1 + 2x_2 + 4x_3$ and $y = y_1 + 2y_2$. This leads to

$$\max_{x \in X} F(x, y) = -x_1 - 2x_2 - 4x_3 - y_1 - 2y_2,$$

where y solves

$$\max_{y \in Y} f(x, y) = 5x_1 + 10x_2 + 20x_3 + y_1 + 2y_2,$$

subject to

$$\begin{aligned} -2x_1 - 4x_2 - 8x_3 - y_1 - 2y_2 &\leq -4, \\ -x_1 - 2x_2 - 4x_3 + 4y_1 + 8y_2 &\leq 8, \\ 2x_1 + 4x_2 + 8x_3 + y_1 + 2y_2 &\leq 16, \\ x_1 + 2x_2 + 4x_3 - 2y_1 - 4y_2 &\leq 4. \end{aligned}$$

The example can now be restated in the form of (4):

$$\max_{x \in X,\ y \in Y} f(y) = y_1 + 2y_2$$

subject to

$$\begin{aligned} -2x_1 - 4x_2 - 8x_3 - y_1 - 2y_2 &\leq -4, \\ -x_1 - 2x_2 - 4x_3 + 4y_1 + 8y_2 &\leq 8, \\ 2x_1 + 4x_2 + 8x_3 + y_1 + 2y_2 &\leq 16, \\ x_1 + 2x_2 + 4x_3 - 2y_1 - 4y_2 &\leq 4, \\ F(x, y) = -x_1 - 2x_2 - 4x_3 - y_1 - 2y_2 &\geq \alpha, \\ x_1 + x_2 + x_3 &\geq \beta. \end{aligned}$$

The first few steps of the algorithm are outlined below. For conciseness, the integer representation of the binary points is used in the discussion. Note that the intermediate results are not unique but depend upon the algorithmic approach used to solve the zero-one subproblems at Steps 1 and 2.

After initialization at Step 0, the algorithm finds at Step 1 the point (4, 3), which is confirmed to be in the inducible region at Step 2. Thus, $\underline{F} = -7$, $S_1^+ = \{3\}$, $S_1^- = \emptyset$, $S_1^0 = \{1, 2\}$, $P_1 = (3)$, $\alpha = -6$, and $\beta = 2$. Solving problem (4) again at Step 1 yields the point (5, 1). At Step 2, $(x^2, y^2) = (5, 3)$,

$F(x^2, y^2) = -8$, so $\underline{F}$ remains at -7. Updating gives $S_2^+ = \{1, 3\}$, $S_2^0 = \{2\}$, $P_2 = (3, 1)$, $\alpha = -6$, and $\beta = 3$. At Step 1 no feasible solution is found, so the algorithm fathoms the current node, goes to Step 6 and backtracks giving $S_2^+ = \{3\}$, $S_2^- = \{1\}$, $S_2^0 = \{2\}$, $P_2 = (3, \underline{1})$, $\alpha = -6$, and $\beta = 0$. Returning to Step 1, the feasible point $(x^3, y^3) = (4, 0)$ is found, and Step 2 again yields $(x^3, y^3) = (4, 3)$ with $F(x^3, y^3) = -7$. At Step 3, the fact that $S_2^0 = \{2\}$ and $x_2^3 = 0$ implies that $J = \emptyset$, and hence, $S_3^+ = S_2^+ = \{3\}$. The parameter β is therefore set to $1 + |S_3^+| = 2$ at Step 5, and control passes to Step 1. The resultant subproblem is infeasible so the algorithm fathoms the current node and jumps to Step 6 and backtracks.

The calculations continue until convergence occurs. The optimal solution for the example is $(x^*, y^*) = (1, 2)$, with $F = -3$. The corresponding tree is shown in Figure 1, where the values of F beside the nodes are those found at Step 2. During execution, the algorithm returns to Step 1 13 times in an attempt to improve upon the incumbent. Subsequently, seven subproblems are solved at Step 2 to find new points in the inducible region. Once again, these results reflect the peculiarities of our specific zero-one code. Finally, if the alternative algorithm is used and problem (4) is completely solved at Step 1, three of these seven subproblems are eliminated at Step 2. Here, the choice of zero-one code is immaterial.

With regard to individual outcomes, note that the leader does better when the variables are restricted to integer, rather than continuous, values. Although this observation cannot be generalized, it arises here because the follower's interests are opposed to those of the leader in the sense that c^2 and d^2 have opposite signs, but the follower's feasible region $\Omega(x)$ (and hence his freedom to advance those interests) is somewhat reduced by the integer requirements.

A second inference that can be drawn from the example concerns the nature of the solution. In particular, note that when compared to the optimum (1, 2),

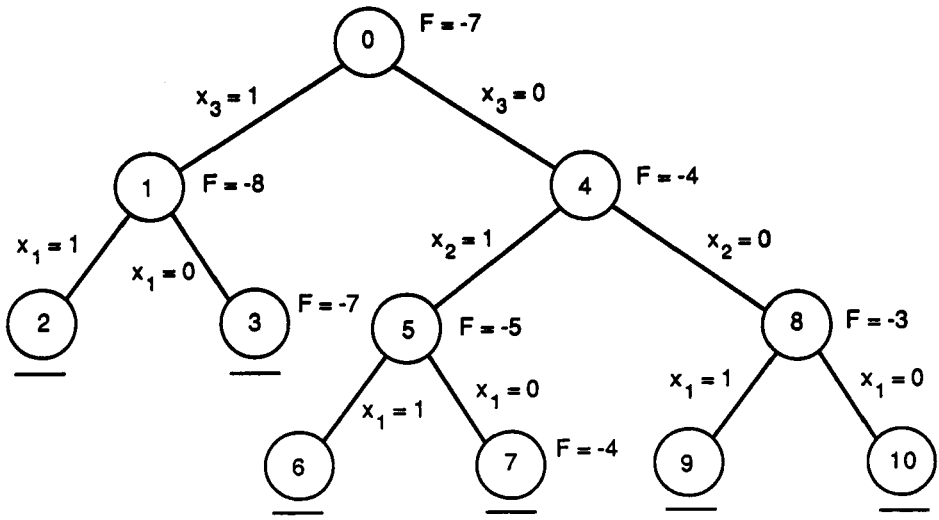

Figure 1. Search tree for example.

the point (2, 0) provides better outcomes for both players but is not in the inducible region. That is, if the leader plays $x = 2$, the follower chooses $y = 2$, thereby degrading the leader's payoff by 1 unit. This verifies, for the integer case, the general fact that solutions to a BLPP are not necessarily Pareto optimal. [If the first term in $f(x, y)$ is dropped, a slight alteration in the problem would be required to demonstrate this result.]

6. COMPUTATIONAL EXPERIENCE

In order to test the performance of the algorithm, a series of problems was randomly generated and solved. In each instance, the constraints had a 0.5 probability of being $\leq$ or $\geq$ inequalities, and all right-hand-side values were positive. The coefficients of the A and B matrices ranged from -10 to 28 with approximately 40% of the nonozero entries being negative. The coefficients of the leader's and follower's objective functions similarly assumed both positive and negative values with the same probability. The number of constraints in each realization was set to 0.4 times the number of variables. Because problem difficulty often depends upon the density of the constraints, two separate cases were examined. The first was characterized by densities of approximately 0.475 for the A and B matrices, and the second by densities of approximately 0.245.

All computations were performed on an IBM 3081-D using the VS Fortran compiler. A Balas-type algorithm similar to the one described by Garfinkel and Nemhauser [10] is used to solve the 0-1 problem at Steps 1 and 2. In their presentation, Garfinkel and Nemhauser suggest several tests that may reduce the enumeration at the expense of additional calculation. After some experimentation with each, only Test 1, which checks for feasibility of a completion, proved to be worthwhile.

In all, 10 problems were run for each case (a case is defined by the way the total number of variables is partitioned among the players). Performance measures include CPU time (sec), the coefficient of variation (standard deviation/mean) for CPU time, the number of nodes in the tree, the node at which the optimal solution is found, and the number of problems examined at Steps 1 and 2.

Table 1 presents the average results for the case where the matrix densities are 0.475. As expected, the CPU time grows exponentially with the size of the problem but seems to be independent of the way the variables are partitioned. Nevertheless, the averages given in Table 1 do not reveal the large differences that were observed among problems of equivalent size. To some extent, this can be seen by examining the coefficient of variation—a measure of the dispersion of CPU time. As an example, the greatest range in CPU time appeared for those problems containing 45 variables where 15 are controlled by the follower. Here, the longest CPU time is 757 sec, while the shortest is less than 3 sec; the standard deviation is 1.47 times the mean. Eliminating these two extremes and redoing the calculations gives an average of 89.6 seconds for the remaining eight problems with a standard deviation to mean ratio of 0.92.

In general, coefficient of variation values near 1 indicate the presence of outliers. This is illustrated by considering the following sample data set {1, 1, 1, 9}. The mean for these four points is 3 and the coefficient of variation is 1.15.

Table 1. Computational results for matrix density of 0.475

No. of variables ($n_1 + n_2$)	Follower variables (n_2)	CPU time (sec)	Coefficient of variation	Iterations		No. of Nodes	Optimal Solution (Node)
				Step 1	Step 2		
15	5	0.063	0.71	26	14	23	13
20	5	0.099	0.58	24	11	25	15
20	10	0.369	1.03	47	29	34	18
25	5	0.507	0.77	39	18	39	26
25	10	1.869	0.79	141	80	119	50
30	5	0.984	0.89	47	19	53	44
30	10	1.228	0.95	44	21	46	24
30	15	3.111	1.21	47	25	43	26
35	5	5.099	1.36	36	14	42	31
35	10	4.965	1.30	49	23	50	41
35	15	13.487	1.15	86	49	71	54
40	5	14.057	1.25	47	15	61	47
40	10	31.874	2.10	52	22	58	45
40	15	18.578	1.22	47	21	51	31
40	20	21.779	1.01	44	21	44	28
45	10	23.076	1.08	20	8	22	16
45	15	147.655	1.47	50	20	7	48
45	20	106.858	0.83	124	64	117	106

This type of pattern was typical of the CPU times accompanying our results. Each problem set contained one, and sometimes two, instances in which the CPU time was at least 1 order of magnitude greater than the majority. Similar observations were made concerning the number of times it was necessary to execute Steps 1 and 2. For problems with 50 or more variables (results not shown in Table 1), computation times often exceeded 900 sec, the cutoff time chosen for the runs.

Table 2 contains the results for the 18 problem sets when the A and B matrix densities are approximately 0.245, or roughly half the value used in the first case. As can be seen, both CPU times and the accompanying variation within the problem set are in general slightly greater for this case. However, the order of magnitude of these two measures is the same. The places where strong differences arise are in the number of times Steps 1 and 2 are executed, the number of nodes in the tree, and the node at which the optimal solution is found. Comparing the last four columns of Tables 1 and 2, we see that these measures may be anywhere from one to two orders of magnitude greater for the less-dense case. The fact that run times are equivalent implies that the less-dense subproblems are much easier and quicker to solve, but are more likely to be feasible. Thus, less fathoming occurs at Step 1 and the search trees grow accordingly. The reverse situation was seen to hold when the matrix densities were increased to about 7.

7. DISCUSSION

After solving approximately 400 randomly generated problems, experience has shown that a wide variation in algorithmic performance exists, even for

Table 2. Computational results for matrix densities of 0.245

No. of variables ($n_1 + n_2$)	Follower variables (n_2)	CPU time (sec)	Coefficient of variation	Iterations		No. of nodes	Optimal solution (node)
				Step 1	Step 2		
15	5	0.337	1.12	180	114	130	11
20	5	2.342	1.13	856	559	591	136
20	10	1.107	0.80	288	183	208	50
25	5	1.338	1.96	377	226	301	89
25	10	4.451	1.50	961	622	676	128
30	5	4.224	2.47	951	534	832	676
30	10	11.395	1.18	2026	1285	1481	400
30	15	8.601	1.15	440	274	329	61
35	5	2.154	0.89	230	108	241	167
35	10	5.224	1.09	636	378	493	240
35	15	11.947	1.02	966	578	775	263
40	5	1.471	0.65	55	21	68	42
40	10	8.602	0.99	148	65	165	110
40	15	35.815	1.22	2424	1456	1832	592
40	20	25.415	0.93	953	573	758	363
45	10	72.553	1.69	2725	1622	2203	755
45	15	110.343	1.50	2497	1473	2047	1064
45	20	184.481	1.24	1093	637	901	294

problems of the same class. And while the number of variables greatly affects overall computation time, little can be predicted in advance for a specific example.

These observations, coupled with the fact that in most instances the algorithm quickly finds good points in the inducible region, suggest a variety of heuristics for speeding termination. These include (1) stopping after a given number of nodes, (2) stopping after a given number of passes through Step 1 or Step 2, or (3) stopping after a given amount of CPU time has elapsed. Justification follows from what appears to be a high correlation between the first and the last four columns in Table 1. A fourth possibility aimed at speeding convergence centers on the choice of branching rules. One alternative that was tried involved ordering the variables according to their coefficient values in the leader's objective function. On average, this produced a small degradation in performance. Likewise, branching on the variables one at a time increased the computational effort anywhere from 50 to 1000 percent.

If any real improvement in the basic algorithm is to be made, though, Step 1 holds the key. Step 1 yields an x^k and a corresponding (i) $y^k \in \Omega(x^k)$ for which $F(x^k, y^k) \geq \alpha$. What we would really like is an x^k and a (ii) $y^k \in \Psi(x^k)$ for which $F(x^k, y^k) \geq \alpha$. Although (i) is necessary for (ii), it provides only weak assurance in general that (ii) might hold. The modified algorithm offered a means of checking for (ii) but the underlying conditions proved to be too weak to be effective. Nevertheless, it might still be possible to sharpen Step 1 to increase the likelihood that y^k is in $\Psi(x^k)$ without having to solve (4) to optimality. A plausible approach is to seek a test for $(x^k, y^k) \in \text{IR}$ based on the degree of harmony or opposition of the two players as measured by the relative sizes of c^2 and d^2, or by the sign of the scalar product $\langle c^2, d^2 \rangle$. This is now being explored.

Finally, it should be mentioned that in repeated testing, the current algorithm always outperformed our more traditional branch-and-bound code [12] designed for the general mixed integer linear BLPP. Not only was the zero-one code able to solve 50% larger problems within the imposed time limits, but it did so in one-tenth to one-half the amount of time. The mixed integer code uses the solution of the LP relaxation (with some qualifications) as a lower bound, and empirically derived branching rules for constructing the tree.

ACKNOWLEDGMENTS

The authors would like to thank the two referees for their suggestions and recommendations for improving the article. Many of their comments were incorporated directly in the text. This work was partially supported by a grant from the Advanced Research Program of the Texas Higher Education Coordinating Board and the Lady Davis Foundation.

REFERENCES

[1] Aiyoshi, E., and Shimizu, K., "Hierarchical Decentralized Systems and Its New Solution by a Barrier Method," *IEEE Transactions on Systems, Man, and Cybernetics*, **SMC-11**(6), 444–449 (1981).

[2] Bard, J.F., and Falk, J.E., "An Explicit Solution to the Multi-Level Programming Problem," *Computers & Operations Research*, **9**(1), 77–100 (1982).

[3] Bard, J.F., and Moore, J.T., "Production Planning with Variable Demand," *Omega*, **18**(1), 35–42 (1990).

[4] Basar, T., and Olsder, G.J., *Dynamic Noncooperative Game Theory*, Academic Press, London, 1982.

[5] Bialas, W.F., and Karwan, M.H., "Two-Level Linear Programming," *Management Science*, **30**(8), 1004–1020 (1984).

[6] Bitran, G.R., "Theory and Algorithms for Linear Multiple Objective Programs with Zero-One Variables," *Mathematical Programming*, **17**, 362–390 (1979).

[7] Candler, W., and Norton, R., "Multi-Level Programming and Development Policy," Working Paper No. 258, World Bank, Washington, DC (1977).

[8] Candler, W., and Townsley, R., "A Linear Two-Level Programming Problem," *Computers & Operations Research*, **9**(1), 59–76 (1982).

[9] Fortuny-Amat, J., and McCarl, B., "A Representation and Economic Interpretation of a Two-Level Programming Problem," *Journal of the Operational Research Society*, **32**(9), 783–792 (1981).

[10] Garfinkel, R.S., and Nemhauser, G.L., *Integer Programming*, Wiley, New York, 1972.

[11] Kiziltan, G., and Yucaoglu, E., "An Algorithm for Zero-One Multiobjective Linear Programming," *Management Science*, **29**(3), 1444–1453 (1983).

[12] Moore, J.T., and Bard, J.F., "The Mixed Integer Linear Bilevel Programming Problem," *Operations Research*, **38**(5), 911–921 (1990).

[13] Roodman, G.M. "Postoptimality Analysis in Zero-One Programming by Implicit Enumerations," *Naval Research Logistics Quarterly*, **19**(3), 435–447 (1972).

[14] Simaan, M., and Cruz, J.B., Jr., "On the Stackelberg Strategy in Nonzero-Sum Games," *Journal of Optimization Theory and Applications*, **11**, 533–555 (1973).

[15] Soyster, A.L., "Inexact Linear Programming with Generalized Resource Sets," *European Journal of Operational Research*, **3**(4), 316–321 (1979).

[16] Villarreal, B., and Karwan, M.H., "Multicriteria Integer Programming: A (Hybrid) Dynamic Programming Recursive Approach," *Mathematical Programming*, **21**, 204–223 (1981).

[17] White, D.J., "A Branch and Bound Method for Multi-Objective Boolean Problems," *European Journal of Operational Research*, **15**(1), 126–130 (1984).
[18] Yu, P.L., "Cone Convexity, Cone Extreme Points, and Nondominated Solutions in Decision Problems with Multiobjectives," *Journal of Optimization Theory and Applications*, **14**(3), 321–377 (1974).

Manuscript received 8/8/87
Manuscript accepted 7/30/91