A Branch and Bound Algorithm for the Bilevel Programming Problem

Article *in* SIAM Journal on Scientific and Statistical Computing · March 1990
DOI: 10.1137/0911017

2 authors, including:

Jonathan Bard
University of Texas at Austin
219 PUBLICATIONS **9,137** CITATIONS

SIAM J. SCI. STAT. COMPUT.
Vol. 11, No. 2, pp. 281-292, March 1990

005

A BRANCH AND BOUND ALGORITHM FOR THE BILEVEL PROGRAMMING PROBLEM*

JONATHAN F. BARD† AND JAMES T. MOORE‡

Abstract. The bilevel programming problem is a static Stackelberg game in which two players try to maximize their individual objective functions. Play is sequential and uncooperative in nature. This paper presents an algorithm for solving the linear/quadratic case. In order to make the problem more manageable, it is reformulated as a standard mathematical program by exploiting the follower's Kuhn-Tucker conditions. A branch and bound scheme suggested by Fortuny-Amat and McCarl is used to enforce the underlying complementary slackness conditions. An example is presented to illustrate the computations, and results are reported for a wide range of problems containing up to 60 leader variables, 40 follower variables, and 40 constraints. The main contributions of the paper are in the step-by-step details of the implementation, and in the scope of the testing.

Key words. bilevel programming, Stackelberg games, branch and bound, complementarity sequential game, linear programming

AMS(MOS) subject classifications. 65-03, 90D05, 90C05

1. Introduction. Hierarchical decision problems involving conflict among the different subunits can often be modeled as a multilevel game. Examples include government regulation, management of a decentralized firm, and the standard max-min problem. In each of these situations, the leader attempts to maximize his objective function by selecting a strategy that anticipates the reactions of the followers. In so doing, if he is limited to influencing rather than controlling subunit outcomes, a Stackelberg game results (see Basar and Selbuz (1979), or Simaan and Cruz (1973)). The bilevel programming problem (BLPP) is a static version of this game where the leader has control over the decision variables $x \in X \subseteq R^{n_1}$, while the follower separately controls the decision variables $y \in Y \subseteq R^{n_2}$ (Aiyoshi and Shimizu (1981), Bard and Falk (1982), Bialas and Karwan (1984)).

In our formulation, it will be assumed that the leader goes first and chooses an x to maximize his objective function $F(x, y)$. The follower then reacts by selecting a y to maximize his individual objective function $f(x, y)$ without regard to the external consequences of his actions. Here, $F: X \times Y \to R^1$ and $f: X \times Y \to R^1$. The problem addressed in this paper is the "linear/quadratic" case given by

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

$$\text{subject to} \quad x \in X = \{x: Dx \geqq d\}, \tag{1b}$$

$$\max_y f(x, y) = c^3 y + x^T Q_1 y + \tfrac{1}{2} y^T Q_2 y, \tag{1c}$$

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

$$y \in Y = \{y: Ey \geqq e\} \tag{1e}$$

* Received by the editors April 13, 1987; accepted for publication (in revised form) March 13, 1989.

† Operations Research Group, Department of Mechanical Engineering, University of Texas, Austin, Texas 78712.

‡ Major, U.S. Air Force, Strategic Air Command Headquarters, Offutt Air Force Base, Omaha, Nebraska 68113.

where A is $m_1 \times n_1$, B is $m_1 \times n_2$, D is $m_2 \times n_1$, E is $m_3 \times n_2$, Q_1 is $n_1 \times n_2$, Q_2 is $n_2 \times n_2$ symmetric, negative semidefinite, and c^1, c^2, c^3, b, d, and e are vectors of conformal dimension. Note that it is always possible to drop components separable in x from the follower's objective function without altering the results. Hence, (1c) does not contain linear and quadratic terms in x.

A significant amount of effort has been devoted to solving (1a)-(1e), which Jeroslow (1985) has shown to be NP-*hard*. A survey can be found in Bialas and Karwan (1984) where they outline both their "high point" algorithm and complementary pivot approach. Also see Candler and Townsley (1982) and Fortuny-Amat and McCarl (1981); for a generalization of (1a)-(1e) to many players see Bard (1983a) and Gardner and Cruz (1978).

The intent of this paper is to describe the implementation and testing of a branch and bound scheme for solving (1a)-(1e) initially suggested by Fortuny-Amat and McCarl. The approach is similar to that developed by Bard (1988) for dealing with the case where $F(x, y)$ is strictly concave, but is more robust as discussed in § 3. In the next section, terminology and assumptions are presented. This is followed in § 3 by the development of the algorithm and an example to highlight its operations; § 4 contains our computational experience and some insights gained from testing. We conclude in § 5 with a discussion of the results.

2. Terminology and assumptions. Two of the basic assumptions underlying bilevel programming are that full information is available to the players, and that cooperation is prohibited. This precludes the use of correlated strategies and side payments.

The following notation is used in the development.

Follower's Rational Reaction Set:

$$M(x) = \{y: y = \operatorname{argmax} [f(x, y): y \in Y, g(x, y) \geqq b]\}.$$

Inducible Region:

$$\mathbb{R} = \{(x, y): x \in X, y \in M(x)\}.$$

In order to ensure that (1a)-(1e) is well posed, we make the additional assumption that the feasible region (1b), (1d), and (1e) is nonempty and compact, and that for each decision taken by the leader, the follower has some room to respond. The rational reaction set $M(x)$ defines this response while the inducible region $\mathbb{R}$ represents the set over which the leader may optimize when given control of all the variables.

Even with the above assumptions, the BLPP may not have a solution. In particular, if $M(x)$ is not single-valued for all permissible x, the leader may not achieve his maximum payoff over $\mathbb{R}$. In order to avoid this situation, it will be assumed that $M(x)$ is a point-to-point map. Because a simple check is available to see if the solution to (1a)-(1e) is unique (see Bard and Falk 1982), we do not feel that this assumption is unduly restrictive. The problem actually solved by our algorithm is $\max \{F(x, y): (x, y) \in \mathbb{R}\}$ without the requirement that $M(x)$ be single-valued.

3. Methodology. Rather than working with (1a)-(1e) in its hierarchical form, we begin by converting it into a standard mathematical program. This is achieved by replacing the follower's problem (1c)-(1e) with his Kuhn-Tucker conditions, and giving control of all the variables to the leader (see Simaan and Cruz (1973)). For $X = \{x: x \geqq 0\}$, $Y = \{y: y \geqq 0\}$, and $m \equiv m_1$, we get

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

$$\text{(2b)} \qquad \text{subject to} \quad x^T Q_1 + y^T Q_2 + u^1 B + u^2 I = -c^3,$$

$$\text{(2c)} \qquad u^1(Ax + By - b) + u^2 y = 0,$$

$$\text{(2d)} \qquad Ax + By \geqq b,$$

$$\text{(2e)} \qquad x, y, u^1, u^2 \geqq 0,$$

where u^1 and u^2 are m- and n_2-dimensional Kuhn-Tucker multipliers, and I is an $n_2 \times n_2$ identity matrix. Constraints (2b), (2c), and (2e) can be interpreted as an explicit representation of the inducible region. Thus, even for the linear/quadratic case, problem (1a)-(1e) is nonconvex and cannot necessarily be solved with a standard nonlinear programming code such as GRG2 (Lasdon et al. (1978)).

As suggested by Fortuny-Amat and McCarl, the basic idea of our algorithm is to suppress the complementarity term (2c) and solve the resulting linear program. At each iteration, a check is made to see if (2c) is satisfied. If so, the corresponding point is in the inducible region, and hence, is a potential solution to (1a)-(1e); if not, a branch and bound scheme is used to implicitly examine all combinations of complementary slackness. It should be mentioned that Fortuny-Amat and McCarl did not implement this scheme but took the more direct approach of replacing (2c) with the following set of inequalities: $u_i \leqq Mz_i$, $g_i \leqq M(1 - z_i)$, where $z_i \in \{0, 1\}$ for $i = 1, \cdots, m + n_2$, and M is a sufficiently large constant. They then solved the resultant problem with a standard zero-one mixed integer code.

Before presenting the algorithm, we introduce some additional notation. Define $u \equiv (u^1, u^2)$, let $W = \{1, \cdots, m + n_2\}$ be the index set for the terms in (2c), and let $\underline{F}$ be the incumbent lower bound on the leader's objective function. At the kth level of the branch and bound tree we define a subset of indices $W_k \subseteq W$, and a path vector P_k (with $|W_k|$ nonzero components) corresponding to an assignment of either $u_i = 0$ or $g_i = 0$ for $i \in W_k$. Now let

$$S_k^+ = \{i: i \in W_k \text{ and } u_i = 0\},$$

$$S_k^- = \{i: i \in W_k \text{ and } g_i = 0\},$$

$$S_k^0 = \{i: i \notin W_k\}.$$

For $i \in S_k^0$, the variables u_i and g_i are free to assume any nonnegative values in the solution of (2a)-(2e) with (2c) omitted, so (2c) will not necessarily be satisfied.

ALGORITHM.

Step 0. (Initialization). Put $k = 0$, $S_k^+ = \varnothing$, $S_k^- = \varnothing$, $S_k^0 = \{1, \cdots, m + n_2\}$, and $\underline{F} = -\infty$.

Step 1. (Iteration k). Set $u_i = 0$ for $i \in S_k^+$ and $g_i = 0$ for $i \in S_k^-$. Attempt to solve (2a)-(2e) without (2c). If the resultant subproblem is infeasible, go to Step 5; otherwise, put $k \leftarrow k + 1$ and label the solution (x^k, y^k, u^k).

Step 2. (Fathoming). If $F(x^k, y^k) \leqq \underline{F}$, go to Step 5.

Step 3. (Branching). If $u_i^k \cdot g_i(x^k, y^k) = 0$, $i = 1, \cdots, m + n_2$, go to Step 4; otherwise, select i for which $u_i^k \cdot g_i(x^k, y^k)$ is largest and label it i_1. Put $S_k^+ \leftarrow S_k^+ \cup \{i_1\}$, $S_k^0 \leftarrow S_k^0 \backslash \{i_1\}$, $S_k^- \leftarrow S_k^-$, append i_1 to P_k, and go to Step 1.

Step 4. (Updating). $\underline{F} = F(x^k, y^k)$.

Step 5. (Backtracking). If no live node exists, go to Step 6. Otherwise branch to the newest live vertex and update S_k^+, S_k^-, S_k^0, and P_k as discussed below. Go to Step 1.

Step 6. (Termination). If $\underline{F} = -\infty$, there is no feasible solution to (1a)-(1e). Otherwise, declare the feasible point associated with $\underline{F}$ the optimal solution.

Step 1 is designed to find a new point that is potentially bilevel feasible. If no solution exists, or the solution does not offer an improvement over the incumbent (Step 2), the algorithm goes to Step 5 and backtracks. At Step 3, a check is made to determine if the complementary slackness conditions are satisfied. In practice, if $|u_i \cdot g_i| < 10^{-6}$, it is considered to be zero. Confirmation indicates that a feasible solution of the bilevel program has been found, and at Step 4, the lower bound on the leader's objective function is updated. Alternatively, if the complementary slackness conditions are not satisfied, the term with the largest product is used at Step 3 to provide the branching variable. Branching is always done on the Kuhn-Tucker multiplier.

At Step 5, the backtracking operation is performed. Note that a live node is one associated with a subproblem that has not yet been fathomed at either Step 1 due to infeasibility or at Step 2 due to bounding, and whose solution violates at least one complementary slackness condition. To facilitate bookkeeping, the path P_k in the branch and bound tree is represented by an l-dimensional vector, where l is the current depth of the tree. The order of the components of P_k is determined by their "level" in the tree. Indices only appear in P_k if they are in either S_k^+ or S_k^- with the entries underlined if they are in S_k^-. Because the algorithm always branches on a Kuhn-Tucker multiplier first, backtracking is accomplished by finding the rightmost nonunderlined component of P_k, underlining it, and erasing all entries to the right. The newly underlined 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.

If we arrive at Step 6 and $\underline{F} = -\infty$, then we conclude that the original constraint region (1b), (1d), (1e) is empty. This will only be the case if the first subproblem at Step 1 is infeasible. Alternatively, the algorithm terminates with the incumbent whose optimality is now established.

PROPOSITION. *Under the uniqueness assumption associated with the rational reaction set $M(x)$, the algorithm terminates with the global optimum of the* BLPP (1a)-(1e).

Proof. The algorithm forces satisfaction of the complementary slackness conditions in problem (2a)-(2e), which is an equivalent representation of (1a)-(1e). By implicitly considering all combinations of $u \cdot g(x, y) = 0$ at Steps 3 and 5, the optimal solution cannot be overlooked. □

Example.

$$\max_x F(x, y) = 8x_1 + 4x_2 - 4y_1 + 40y_2 + 4y_3 \quad \text{where } y \text{ solves}$$

$$\max_y f(x, y) = -x_1 - 2x_2 - y_1 - y_2 - 2y_3,$$

$$\begin{aligned} \text{subject to} \quad & y_1 - y_2 - y_3 \geqq -1, \\ -2x_1 + {} & y_1 - 2y_2 + 0.5y_3 \geqq -1, \\ -2x_2 - 2y_1 + {} & y_2 + 0.5y_3 \geqq -1, \\ x \geqq 0, \quad & y \geqq 0. \end{aligned}$$

This example was taken from Candler and Townsley (1982). When it is rewritten in its equivalent form (2a)-(2e), six Kuhn-Tucker multipliers appear, implying that it may be necessary to solve as many as $2^7 - 1 = 127$ subproblems before terminating. In fact, the optimal solution was uncovered on the fourth iteration but was not confirmed until 10 subproblems were examined.

More specifically, after initializing the data, the algorithm finds a feasible solution to the Kuhn-Tucker representation with the complementary slackness conditions omitted, and proceeds to Step 3. The current point, $x^1 = (0, 0)$, $y^1 = (1.5, 1.5, 1)$, $u^1 = (0, 0, 0, 1, 1, 2)$, with $F(x^1, y^1) = 58$ does not satisfy complementarity so a branching variable is selected (u_6) and the index sets are updated, giving $S_1^+ = \{6\}$, $S_1^- = \varnothing$, $S_1^0 = \{1, 2, 3, 4, 5\}$, and $P_1 = (6)$. In the next two iterations, the algorithm branches on u_5 and u_4, respectively. Now, three levels down in the tree, the current subproblem at Step 1 turns out to be infeasible, so the algorithm goes to Step 5 and backtracks. The index sets are $S_3^+ = \{5, 6\}$, $S_3^- = \{4\}$, and $S_3^0 = \{1, 2, 3\}$, and $P_3 = (6, 5, \underline{4})$. Going to Step 1, a feasible solution is found that passes the test at Step 2 and satisfies the complementary slackness conditions at Step 3. Continuing at Step 4, $\underline{F} = 29.2$. Backtracking at Step 5 yields $S_4^+ = \{6\}$, $S_4^- = \{5\}$, and $S_4^0 = \{1, 2, 3, 4\}$, and $P_4 = (6, \underline{5})$. Returning to Step 1, another feasible solution is found, but at Step 2, the value of the leader's objective function is less than the incumbent lower bound, so the algorithm goes to Step 5 and backtracks, giving $S_5^+ = \varnothing$, $S_5^- = \{6\}$, $S_5^0 = \{1, 2, 3, 4, 5\}$, and $P_5 = (\underline{6})$. The calculations continue until no live vertices exist. The optimal solution is $x^* = (0, 0.9)$, $y^* = (0, 0.6, 0.4)$, $u^* = (0, 1, 3, 6, 0, 0)$ with $F^* = 29.2$. The branch and bound tree for this example is shown in Fig. 1.

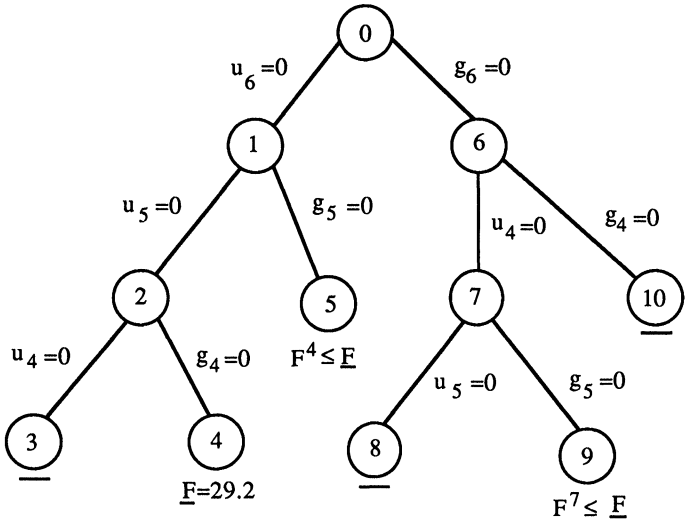

FIG. 1. *Branch and bound tree for example.*

By way of comparison, when this problem was solved with the separable programming approach of Bard and Falk (1982), the optimal solution was uncovered on the 51st iteration but not recognized until iteration 103. (Each iteration required the solution of a linear program in $n_1 + n_2 + 2m$ variables and $2(m + n_2) + 1$ constraints.) This result typifies the relative performance of these two algorithms.

Finally, we note that the above procedure is considerably more general than that proposed by Bard (1988). Although both use branch and bound concepts, the latter takes an active set approach, adhering to the inducible region until a local optimum is found. This requires more bookkeeping, and will only succeed if the leader's objective function is strictly concave (for a maximization problem). Alternatively, our algorithm would easily solve this version of the BLPP if an appropriate nonlinear optimization

code were used at Step 1 for the subproblems (see Edmunds (1988) for an implementation).

4. Computational experience. In order to test the efficiency of the algorithm, a series of problems was randomly generated and solved. For the primary cases reported, the coefficients of the A, B, Q_1, and Q_2 matrices ranged between -15 and 45 with approximately 25% of the entries being less than zero. Each matrix had a density of about 0.4. The coefficients of the two objective functions varied between -20 and 20 with approximately 50% being less than zero. The number of constraints in each problem was set at 0.4 times the total number of variables, and the right-hand side (RHS) values ranged between 0 and 50. The signs of the constraints had a 0.7 probability of being $\leqq$ and a 0.3 probability of being $\geqq$.

These settings are somewhat arbitrary, but were chosen to be consistent with previous work (e.g., see Bard (1983b), Bialas and Karwan (1984)). The advantage of having all coefficients of equal magnitude is that the resultant problems are almost always stable numerically. From a testing point of view, the matrix density factor plays an important role in generating random problems. Depending on the number of variables and constraints in the model, if this value is set too low a considerable amount of work may be required to assure that each realization is usable; i.e., unbiased and feasible with at least one nonzero element in each row and column. For our generator, a value of 0.4 was sufficient to guarantee usability in all but a few instances.

To complement the primary runs, additional testing was done on a subset of the original problems. In the first case, we investigated the relationship between algorithmic performance and the density of the A and B matrices. Here, the range of coefficient values remained the same. In the second case, an attempt was made to construct "ill-conditioned" problems by selectively generating coefficients on the order of 10^6.

All computations were performed on an IBM 3081-D using the VS Fortran compiler. As now coded, the subproblems encountered at Step 1 are solved with the linear programming (LP) subroutine library XMP (Marsten (1981)); however, any LP package including those based on interior point methods could be used. Multiplier values required to be zero on a given iteration are accommodated by fixing their upper and lower bounds at zero. Similarly, constraints required to be binding are satisfied by setting their slacks to zero. Both of these operations are easily handled in XMP by a subroutine call. All variable bounds are treated implicitly, so additional constraints are not needed in the formulation.

4.1. Results. Table 1 summarizes our computational experience with the algorithm for the all linear case; i.e., $f(x, y) = c^3 y$. Each entry represents the average of 10 randomly generated problems. In all, 110 problems were solved ranging in size from 40 to 100 variables, and 16 to 40 constraints. Performance measures include CPU time (seconds), the number of nodes in the branch and bound tree, and the node at which the optimal solution is found. Also, data for the largest and smallest search trees are given as a measure of variability.

As expected, the CPU time grows exponentially with the size of the problem, but more importantly, depends on the way the variables are partitioned between the players. As the number of variables controlled by the follower increases, the number of variables included in the branch and bound process increases along with the expected computational burden. Compare, for example, the two cases with 90 variables (and 36 constraints). On average, 81 additional CPU seconds are required for the case where $n_2 = 45$.

Also, as seen in Table 1, large differences in computational effort often result among problems of equivalent size. For the 100 variable case, 34 subproblems had to

TABLE 1
Computational results for the all linear case.

No. of variables (n_1+n_2)	Follower variables (n_2)	No. of consts. (m)	CPU time (sec)	Average no. of nodes	No. of nodes (range)	Optimal solution (node)
40	12	16	4.44	18	6-42	11
40	16	16	8.50	40	8-202	27
50	15	20	17.24	39	10-112	26
50	20	20	16.46	35	18-124	23
50	25	20	32.39	73	16-218	46
50	30	20	179.01	447	30-1250	391
70	28	28	106.99	96	32-270	67
70	35	28	122.26	106	26-246	84
90	36	36	352.37	138	30-384	81
90	45	36	433.14	185	48-374	122
100	40	40	294.22	159	34-476	120

be solved at one extreme and 476 at the other. The corresponding CPU times were 85 seconds and 804 seconds, respectively. In general, the optimum is not uncovered until 60 to 70% of the realized tree is examined. This implies that if the algorithm is stopped prematurely the best solution might be missed. A final point to be made about the empirical results is that about 45% of the nodes in the search tree are fathomed due to infeasibility, and rarely (only 5% of the time) due to a solution being in the inducible region. As discussed in § 4.2, this is due in large part to the branching rule employed at Step 3. The remaining 50% are fathomed at Step 2 when the relaxed solution is less than or equal to the incumbent.

In order to see if the algorithm performed any differently when the follower's objective function contained quadratic terms, the problem sets were rerun for the case where $f(x, y) = c^3y + x^TQ_1y + \frac{1}{2}y^TQ_2y$. In the actual implementation, Q_2 is coded as a lower triangular matrix to facilitate data input. This format eliminates duplicate entries.

The second set of results is presented in Table 2 where little if any significant difference can be seen when compared to the results in Table 1. Nevertheless, the algorithm does seem to take a bit longer to converge when the quadratic terms are

TABLE 2
Computational results for the linear/quadratic case.

No. of variables (n_1+n_2)	Follower variables (n_2)	No. of consts. (m)	CPU time (sec)	Average no. of nodes	No. of nodes (range)	Optimal solution (node)
40	12	16	4.83	20	6-55	13
40	16	16	9.01	45	7-238	30
50	15	20	16.79	37	9-114	25
50	20	20	18.71	41	22-139	31
50	25	20	38.28	82	19-205	48
50	30	20	192.37	482	30-1163	372
70	28	28	116.99	104	38-299	75
70	35	28	112.65	104	32-281	71
90	36	36	393.21	143	28-407	89
90	45	36	451.78	202	53-392	130
100	40	40	363.93	178	38-511	132

added. Only problem sets 3 and 8 show an improvement. In general, this slight degradation in performance was traced to the fact that (2b) was more easily satisfied when terms in x and y were present. Compared to the all linear case, fathoming at Step 1 due to infeasibility was not as likely to occur at the early iterations. This produced slightly larger trees.

The next set of runs was designed to see how algorithmic performance varied with the density of the A and B matrices. In order to limit the computational effort of this phase of the analysis, two representative problems sets were singled out for investigation, and only the linear case was considered. The first problem set is characterized by parameter values $(n, m, n_1, n_2) = (50, 20, 25, 25)$, and the second set by $(70, 28, 35, 35)$. The results for density factors of 0.2, 0.3, and 0.4 are reported in Table 3. All entries represent an average of 10 cases. As the density is reduced from 0.4 to 0.3, the average CPU time stays about the same while the number of nodes in the search tree increases modestly. For a density factor of 0.2, however, a significant drop in CPU time is observed. In the first case, this is accompanied by an incresse in the average size of the tree from 73 nodes to 100 nodes, and in the second case by a decrease from 106 nodes to 82 nodes.

TABLE 3
Results for different matrix densities for all linear case.

No. of variables (n_1+n_2)	Follower variables (n_2)	No. of consts. (m)	CPU time (sec)	Average no. of nodes	No. of nodes (range)	Optimal solution (node)
Matrix density = 0.4						
50	25	20	32.39	73	16-218	46
70	35	28	122.26	106	26-246	84
Matrix density = 0.3						
50	25	20	32.11	86	16-208	68
70	35	28	135.52	119	26-332	95
Matrix density = 0.2						
50	25	20	22.29	100	22-382	59
70	35	28	67.84	82	16-236	55

Although no definitive conclusions should be drawn from these findings, it would be fair to say that a positive correlation exists between the overall density of the problem and the average time spent on each subproblem. This might be accounted for by a combination of factors. First, as the density decreases, the average time XMP takes to solve each subproblem decreases as well. Second, lower densities increase the likelihood that subproblems at a given level in the tree will be infeasibility. This implies that fewer LPs will have to be solved.

Finally, it should be mentioned that when the density factor was set to 0.1, our random problem generator failed to produce any problems in the 50 variable case that did not have either a null row or null column (most of these problems were feasible, though). In the 70 variable case, about 1 in 20 randomly generated problems were usable.

The last set of runs was aimed at determining how well the algorithm performs on problems with widely varying coefficients. In all, eight different scenarios were examined for the 50 variable case. The density factor was held constant at 0.4 and no quadratic terms were included in the follower's objective function $f(x, y)$. Each scenario was characterized by coefficient values randomly selected from one of two ranges. For

the cost coefficients, the ranges were $[-20, 20]$ and $[-10^6, 10^6]$; for the A and B matrices, the ranges were $[-15, 45]$ and $[-2.5 \times 10^5, 7.5 \times 10^5]$; and for the RHSs, the ranges were $[0, 50]$ and $[0, 10^6]$.

The results are displayed in Table 4. Each row represents an average of 10 cases, with the data in the first row taken from Table 1. Average CPU times range from 23.4 seconds to 78.5 seconds, but no clear patterns emerge. However, the algorithm does seem to be slightly less efficient when the problems are ill-conditioned. The only real point worth making is that XMP had no trouble solving any of the subproblems.

TABLE 4
Results for ill-conditioned problems.
($n = 50$, $m = 20$, $n_1 = n_2 = 25$, density = 0.4).

Range of coefficients†			CPU	Average no. of	No. of nodes	Optimal solution
Matrices	Cost	RHS	time	nodes	(range)	(node)
[−15, 45]	[−20, 20]	[0, 50]	32.4	73	16–218	46
[−15, 45]	[−20, 20]	[0, e6]	43.6	119	14–284	109
[−15, 45]	[−e6, e6]	[0, 50]	78.5	207	60–420	97
[−15, 45]	[−e6, e6]	[0, e6]	34.6	100	30–216	73
[−0.25e6, 0.75e6]	[−20, 20]	[0, 50]	45.9	102	26–242	83
[−0.25e6, 0.75e6]	[−20, 20]	[0, e6]	51.8	141	18–264	110
[−0.25e6, 0.75e6]	[−e6, e6]	[0, 50]	23.4	56	24–82	41
[−0.25e6, 0.75e6]	[−e6, e6]	[0, e6]	47.6	132	52–194	102

† The notation e6 denotes 10^6.

It is interesting to compare the above findings with those reported by others. Although Fortuny-Amat and McCarl (1981) did not seriously test their scheme, it is an easy matter to obtain an assessment. After investigating a few 20 variable problems with ZOOM (a zero-one, mixed integer version of XMP), we found that the accompanying run times and search trees were 10 to 100 times larger than ours. The reasons for this were twofold. First, their approach leads to problems with an additional $2(m + n_2)$ constraints and $m + n_2$ variables; second, ZOOM has its own built in branching rules that are not necessarily "optimal" for BLPPs (the same could be said for any commercial mixed integer code).

Bialas and Karwan (1984) report results for both their "Kth-best" algorithm and their Parametric Complementary Pivot (PCP) approach. (While the former did not fare too well and will not be discussed, the latter should really be considered a heuristic because convergence is not guaranteed. In addition, it is limited by the requirement that the leader's objective function coefficients associated with the follower's variables be nonnegative; i.e., $c^2 \geqq 0$.) The largest problems they solved contained 50 variables, with 20 being controlled by the follower. The number of constraints was fixed at 0.4 times the number of variables. Each data set contained five problems, and all computations were done on a CDC Cyber 174 using a Fortran IV code.

Table 5 presents a comparison of the PCP algorithm, our branch and bound scheme, and the separable approach of Bard and Falk. The latter is roughly equivalent to the zero-one formulation of Fortuny-Amat and McCarl in that both approaches lead to problems of nearly identical size and structure. Note that Bard's (1983b) grid search algorithm is not discussed because it only works for BLPPs whose solutions are known to be Pareto-optimal. As can be seen, the first two algorithms are on equal footing with respect to CPU time, and outdistance the third by more than an order of

TABLE 5
Comparison with other algorithms.

No. of variables (n_1+n_2)	Follower variables (n_2)	PCP CPU time (sec)†	B&B CPU time (sec)‡	Separable CPU time (sec)‡
40	12	4.46	4.44	76.25
40	16	11.23	8.48	145.93
50	15	16.48	17.24	298.31
50	20	16.52	16.46	344.77

† Average of 5 problems; CDC Cyber 174.
‡ Average of 10 problems; IBM 3081-D.

magnitude. Note that the Cyber 174 and the IBM 3081-D each perform about 1.7 million floating point operations per second when solving dense systems of linear equations using the LINPACK software (Dongarra (1986)). Nevertheless, the experimental nature of the codes, coupled with the fact that different test problems and different machines were used in the analyses, strongly argue against drawing all but the most tentative conclusions from the data in Table 5. To credibly determine the relative performance of each algorithm, a much more deliberate experimental design would have to be established. At a minimum, it would be necessary to examine a wide variety of problems under identical conditions using the same LP package at each stage in the computations.

4.2. Alternatives explored during testing. In the actual implementation, whenever a feasible solution is found at Step 1, the accompanying basis is stored in XMP format and used as the starting point for the next subproblem. Because this subproblem only differs from its parent by a single constraint (at the next level down just one additional u_i or g_i is set to zero), only a few pivots are normally required to regain feasibility. This was seen to provide a relative advantage when compared to two other procedures tested for maintaining a starting basis. For the first alternative, the previous basis whether feasible or not was used to initiate the next subproblem. For small formulations the results suggested no significant difference in CPU time; but for problems with 50 or more variables, a slight degradation in performance was observed. The second alternative used the basis associated with the last point found in the inducible region. If none existed, the last basis was used. This procedure yielded CPU times two to three times larger than the others and hence is not recommended. In general, the ability to quickly find a feasible solution is the key. On average, the latter procedure spent more than half its time in phase I of the simplex algorithm.

An important determinant of computational efficiency is the rule for selecting the branching variable at Step 3. The rule chosen branches on the Kuhn-Tucker (KT) multiplier associated with the largest complementarity term $u_i \cdot g_i$. Another rule that was examined, primarily because it worked well when nonlinear BLPPs were solved with an active set strategy (see Bard (1988)), proved to be noncompetitive. In this case, the selection is made by finding the surface on which F increases most rapidly; i.e., by solving

$$\max_i \left[\nabla F(x^k, y^k) \cdot \nabla g_i(x^k, y^k) / \|\nabla F(x^k, y^k)\| \, \|\nabla g_i(x^k, y^k)\|\right] \tag{3}$$

where ∇ is the gradient operator and $\|\cdot\|$ is the Euclidean norm. Using this rule and branching on g produced large increases in CPU time; branching on u led to some

relative improvement but anywhere from 50 to 100 additional subproblems had to be solved. These results were observed for moderate sized problems where $n_1 = 15$, $n_2 = 10$, and $m = 10$. Note that for linear/quadratic BLPPs, (3) automatically establishes the branching order. This is probably the reason why it did not work well.

After testing several other rules, it became evident that it is never advantageous to branch on g first. In general, when this strategy is used there is little early fathoming due to infeasibility so the left-hand side of the tree (denoted by $g_i = 0$) grows rapidly. Alternatively, by forcing the KT multipliers to zero, one of two situations quickly arises: either the gradient equations (2b) become infeasible or a point in the inducible region emerges. For this reason, we limited our testing to the following branching rules:

(1) KT multiplier associated with largest $u \cdot g$ product,
(2) KT multiplier with largest value,
(3) KT multiplier with smallest value,
(4) KT multiplier associated with largest g,
(5) KT multiplier associated with smallest g,
(6) KT multiplier associated with smallest $u \cdot g$ product,

where the first is the procedure implemented. Table 6 displays our findings when each of these rules is applied to the original 50 variable problem set. The data represent an average of 10 cases for the all linear version of problem (1a)-(1e).

TABLE 6
Comparison of results for different branching rules.
($n = 50$, $m = 10$, $n_1 = 25$, $n_2 = 25$, density = 0.4)

Rule no.	CPU time	Average no. of nodes	No. of nodes (range)	Optimal solution (node)
1	32.39	73	16-218	46
2	38.91	78	15-225	49
3	98.44	127	20-404	83
4	72.12	106	18-374	71
5	47.66	82	16-271	57
6	68.83	92	18-326	64

As shown in Table 6, none of the variants performed as well as the first rule. Increases in average CPU time of roughly 5 to 66 seconds or 12 to 230% can be observed. In general, those rules that give priority to large values of u seem to do better; the first two rules exhibit almost identical performance.

One possible explanation for these results centers on (2b). Here, the second set of KT multipliers u^2, corresponding to the nonnegativity constraints on the y variables, can be viewed as slacks. At the early stages of the computations these variables tend to take on large values and are hence selected for branching. When this happens, (2b) is usually more difficult to satisfy so fathoming due to infeasibility is more prevalent.

By implication, a rule that gives priority to the u^2 variables might also be a good choice, but this remains to be tested. Finally, note that the results accompanying this phase of the analysis might very well have been different had quadratic terms been included in follower's objective function.

5. Summary and conclusions. Experience has shown that even for the simplest of formulations, the bilevel programming problem is inherently difficult to solve. The branch and bound algorithm developed in this paper attempts to exploit some of the

special structure in the problem, and in so doing, is able to achieve rapid convergence. Linear problems with up to 100 variables and 40 constraints can be solved in less than 300 seconds on average. After thorough testing, the algorithm's performance and robustness are seen to compare favorably with virtually all contenders. A scarcity of data on other algorithms, though, limits the strength of this assertion. More and better controlled experiments are needed before any final conclusions can be drawn.

Nevertheless, one of the main advantages of the branch and bound approach is that it is quite general. Although our analysis centered on the linear/quadratic formulation, solutions to the nonlinear problem can be readily obtained by substituting a more general code such as GRG2 for the XMP library. The algorithm is valid for all functional forms, as well as the case where more than one follower is present. Convergence to the global optimum, though, can only be guaranteed when certain convexity and separable properties hold.

REFERENCES

E. AIYOSHI AND K. SHIMIZU (1981), *Hierarchical decentralized systems and its new solution by a barrier method*, IEEE Trans. Systems, Man, Cybernetics, 11, pp. 444–449.

J. F. BARD (1988), *Convex two-level optimization*, Math. Programming, 40, pp. 15–27.

——— (1983a), *Coordination of a multidivisional firm through two levels of management*, Omega, 11, pp. 457–468.

——— (1983b), *An efficient point algorithm for a linear two-stage optimization problem*, Oper. Res., 31, pp. 670–684.

J. F. BARD AND J. E. FALK (1982), *An explicit solution to the multi-level programming problem*, Comput. Oper. Res., 9, pp. 77–100.

T. BASAR AND H. SELBUZ (1979), *Closed loop Stackelberg strategies with applications in optimal control of multilevel systems*, IEEE Trans. Automat. Control, 24, pp. 166–178.

W. F. BIALAS AND M. H. KARWAN (1984), *Two-level linear programming*, Management Sci., 30, pp. 1004–1020.

W. CANDLER AND R. TOWNSLEY (1982), *A linear two-level programming problem*, Comput. Oper. Res., 9, pp. 59–76.

J. J. DONGARRA (1986), *Performance of various computers using standard linear equations software in a Fortran environment*, Technical Memorandum No. 23, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL.

T. A. EDMUNDS (1988), *Algorithms for nonlinear bilevel mathematical programs*, Ph.D thesis, Department of Mechanical Engineering, University of Texas, Austin, TX.

J. FORTUNY-AMAT AND B. MCCARL (1981), *A representation and economic interpretation of a two-level programming problem*, J. Oper. Res. Soc., 32, pp. 783–792.

B. F. GARDNER, JR. AND J. B. CRUZ, JR. (1978), *Feedback Stackelberg strategy for M-level hierarchical games*, IEEE Trans. Automat. Control, 23, pp. 489–491.

R. G. JEROSLOW (1985), *The polynomial hierarchy and a simple model for competitive analysis*, Math. Programming, 32, pp. 146–164.

L. S. LASDON, A. D. WARREN, A. JAIN, AND M. RATNER (1978), *Design and testing of a generalized reduced gradient code for nonlinear programming*, ACM Trans. Math. Software, 4, pp. 34–50.

R. E. MARSTEN (1981), *The design of the* XMP *linear programming library*, ACM Trans. Math. Software, 7, pp. 481–497.

M. SIMAAN AND J. B. CRUZ, JR. (1973), *On the Stackelberg strategy in nonzero-sum games*, J. Optim. Theory Appl., 11, pp. 533–555.