Journal of Global Optimization **8**: 217–233, 1996.
© 1996 *Kluwer Academic Publishers. Printed in the Netherlands.*

A Hybrid Tabu-Ascent Algorithm for the Linear Bilevel Programming Problem

M. GENDREAU, P. MARCOTTE AND G. SAVARD marcotte@iro.umontreal.ca
DIRO, Université de Montréal, Montréal, Canada H3C 3J7

Received December 22, 1994; Revised August 25, 1995

Abstract. The linear Bilevel Programming Problem (BLP) is an instance of a linear hierarchical decision process where the lower level constraint set is dependent on decisions taken at the upper level. In this paper we propose to solve this NP-hard problem using an adaptive search method related to the Tabu Search metaheuristic. Numerical results on large scale linear BLPs are presented.

Keywords: Bilevel programming, adaptive search methods, combinatorial optimization, Tabu Search

1. Introduction

In this paper we address the numerical solution of the linear bilevel problem (BLP) that consists in finding vectors x and y maximizing the linear form $c_1x + d_1y$, under the constraint that y be optimal for the lower level program [1] :

$$\begin{aligned} \max_z \quad & d_2z \\ \text{subject to} \quad & Ax + Bz \leq b \\ & z \geq 0, \end{aligned}$$

where A is an $m \times n_x$ matrix and B an $m \times n_y$ matrix. The BLP is sometimes recorded in the following format:

$$\begin{aligned} \max_x \quad & c_1x + d_1y \\ \max_y \quad & d_2y \\ \text{subject to} \quad & Ax + By \leq b \\ & x, y \geq 0. \end{aligned} \tag{1}$$

The feasible region $P = \{(x, y) | Ax + By \leq b, x, y \geq 0\}$ is, for simplicity, assumed to be nonempty and bounded. Upon introduction of the multifunction $R(x)$ that associates to a vector x the set of optimal y-solutions to the lower level problem, i.e.,

$$R(x) = \arg\max_y \{d_2y | By \leq b - Ax, y \geq 0\},$$

the BLP can be rewritten as

$$\begin{array}{ll} \max\limits_{x,y} & c_1 x + d_1 y \\ \text{subject to} & Ax + By \leq b \\ & x \geq 0 \\ & y \in R(x). \end{array}$$

Some sets play an important role in the theory of bilevel programming. We denote by R the set of x-vectors for which the set $R(x)$ is nonempty, and by $D = \{\lambda B \geq d_2, \lambda \geq 0\}$ the feasible set associated with the dual of the lower level linear problem. Notice that D does *not* depend on the upper level vector x. The set $\{(x, y) \in P | y \in R(x)\}$ is usually referred to as the *induced region*.

This instance of a linear hierarchical problem was introduced by Candler and Townsley [6], Bard and Falk [2] and Bialas and Karwan [5], who solved small test problems. Jeroslow [12] showed that the BLP is NP-hard, while Vicente, Savard and Júdice [19] proved that the problem of obtaining a certificate of local optimality for the BLP is also strongly NP-hard. Recently, Savard and Gauvin [17] proposed a method for finding a local descent direction for nonlinear bilevel programs. When applied to the BLP, the worst-case complexity of their algorithm is exponential, unless the polyhedron P is nondegenerate. Well-known problems in combinatorial optimization, such as the general linearly-constrained concave quadratic programming problem, or instances of hard mixed-integer problems, can be polynomially reduced to the BLP (see Hansen, Jaumard and Savard [10]).

Recently, some hope of solving larger problems has been raised, following the works of Bard and Moore [4], Júdice and Faustino [13] and Hansen, Jaumard and Savard [10]. The methods proposed by these authors (branch-and-bound, variable elimination, parametric complementarity pivoting) are of a combinatorial nature and yield exponential-time algorithms. One aim of the present paper is to extend the range of linear BLPs solvable to near-optimality by heuristic methods whose computational growth rate is low. The proposed methodology involves a local search akin to the Tabu Search introduced by Glover [7] [8] and under the terminology "steepest ascent, mildest descent", by Hansen [9]. The idea of using local search techniques for solving the BLP is not new. For example, Mathieu, Pittard and Anandalingam [15] have proposed an adaptation of the simulated annealing technique to the linear BLP. However, their approach could not address the large problem instances dealt with in the current paper. A good bibliographic survey of linear and nonlinear bilevel programming can be found in Vicente and Calamai [18].

A side result of our analysis is the development of a primal-dual startup procedure which performs exceptionally well. Frequently, the search phase could not improve on the initial solution produced by the startup procedure. Whenever this favorable situation occurred, we tried to check, using the exact algorithm of Hansen, Jaumard and Savard [10], whether this initial solution was optimal. In most cases, it was.

The paper is organized as follows: Section 2 is devoted to a description of the algorithm, including the startup procedure; Section 3 presents extensive numerical tests; Section 4 concludes the paper.

2. The algorithm

The algorithm is composed of three main building blocks: a *startup phase*, designed to produce a good initial solution, a *local ascent* phase and a *Tabu* phase, whose aim is to move away from and improve on the current locally optimal solution, whenever possible. Each of these phases is described in more detail below.

2.1. The initialization procedure

For a given x in R, a vector y is optimal for the lower level problem if and only if it satisfies, together with a dual vector λ:

$$\begin{aligned} By &\le b - Ax && \text{primal} \\ y &\ge 0 && \text{feasibility} \\[1ex] \lambda B &\ge d_2 && \text{dual} \\ \lambda &\ge 0 && \text{feasibility} \\[1ex] (\lambda B - d_2)y &= 0 && \text{complementary} \\ \lambda(b - Ax - By) &= 0 && \text{slackness.} \end{aligned}$$

If one substitutes to the lower level program the above primal-dual optimality conditions, one obtains the equivalent one-level formulation

$$\begin{aligned} \max_{x,y,\lambda} \quad & c_1x + d_1y \\ \text{subject to} \quad & Ax + By \le b \\ & x, y \ge 0 \\ & \lambda B \ge d_2 \\ & \lambda \ge 0 \\ & (\lambda B - d_2)y = 0 \\ & \lambda(b - Ax - By) = 0. \end{aligned}$$

As in Anandalingam and White [1], we can penalize the complementarity slackness terms to obtain the linearly constrained single-level program

$$\begin{aligned} \max\nolimits_{x,y,\lambda} \quad & c_1x + d_1y - M[(\lambda B - d_2)y + \lambda(b - Ax - By)] \\ & = c_1x + d_1y + Md_2y - M\lambda(b - Ax) \\ \text{subject to} \quad & Ax + By \le b \end{aligned} \tag{2}$$

$$\lambda B \geq b_2.$$
$$x, y, \lambda \geq 0$$

Whenever M is sufficiently large, the penalty is exact in the sense that problem (2) and the original BLP (1) admit the same solution sets. If an optimal dual vector λ were known a priori, (2) would reduce to a standard linear program. Our heuristic procedure estimates λ in the simplest fashion by setting it, for a given upper level vector x, to an optimal dual solution of the lower level problem. The penalized problem is then solved with respect to the x and y-variables. If the resulting vector y is not in $R(x)$, the penalty parameter M is increased, and the procedure repeated. The rationale behind this strategy is to generate a sequence of solutions converging to an (x, y)-solution that satisfies the condition $y \in R(x)$, while favoring the upper level's objective. This parametric scheme is reminiscent of Bard's efficient point algorithm [3] which solves the bicriterion problem

$$\begin{array}{ll} \max\limits_{x,y\geq 0} & c_1x + d_1y + Md_2y \\ \text{subject to} & Ax + By \leq b \end{array}$$

for increasing values of the parameter M, until a rational point (x, y) with y in $R(x)$ is identified. This program is akin to (2). However this strategy always keeps away from non Pareto-optimal maxima, either local or global (see Haurie, Savard and White [11] for a discussion of this topic). This is due to the absence of the correcting term $M\lambda(b - Ax)$ of (2), and may lead to bad feasible solutions.

Our initialization algorithm, called Algorithm INIT, is described below:

ALGORITHM INIT

```
Step 0: Select a parameter $\Delta M$ and set $M$ to 0.

Step 1: Let $(x(M), y(M)) \in \arg\max_{(x,y)\in P} \{c_1x + d_1y + M[d_2y - \lambda(M)(b - Ax)]\}$

Step 2: Let $\lambda(M) \in \arg\min_{\lambda\in D} \{\lambda(b - Ax(M))\}$.

Step 3: if $d_2y(M) - \lambda(M)(b - Ax(M)) = 0$ then go to Step 4
                                      else set $M = M + \Delta M$ and GOTO Step 1.

Step 4: Output the solution $(x(M), y(M))$.
```

□

At Step 2 of the algorithm, one solves the linear program

$$\begin{aligned} \min_{y} \quad & d_2 y \\ \text{subject to} \quad & By \le b - Ax(M) \\ & y \ge 0. \end{aligned}$$

and obtains an optimal primal solution $y(x(M))$ as well as the optimal dual vector $\lambda(M)$.

Some variants of Algorithm INIT have been implemented, but proved less effective, and were not retained. For instance, one could record the feasible solution $(x(M), y(x(M)))$ yielding the highest objective value $c_1 x(M) + d_1 y(x(M))$. This solution might *not* be the one corresponding to the highest (last) value of the penalty parameter M considered. One could also try, at a somewhat higher computational cost, to solve more accurately the bilinear program corresponding to a given value of the penalty parameter M. One should be aware, however, that this subproblem is theoretically as hard to solve as the original bilevel program, and that one should satisfy oneself with a local maximum. Such a local maximum could be obtained, for example, by iteratively solving linear programs with respect to the (x, y) and λ vectors, à *la* Gauss-Seidel.

The initialization phase is then completed by a local search step where pivots in (x, y)-space that improve the leader's objective while leaving y in the induced region $R(x)$ are performed, until no (local) improvement can be found in this manner. This procedure could halt before a local maximum is actually identified, since degenerate pivots are *not* considered. If one demands that a local maximum be achieved, one should be ready to explore all neighbors of the current vertex, of which there could be exponentially many.

Remark: The lower level problem is generically highly degenerate. Indeed, the optimal solution of BLP occurs at an extreme point of P. Consequently, a basic solution vector y of the lower level problem must have at least k zero components, where k is the number of nonzero components of the upper level decision vector, including the slack variables associated with the constraints $Ax + By \le b$. □

2.2. The Tabu phase

The aim of the Tabu phase of the algorithm is, starting at a point (x^0, y^0) on the induced region, to determine another point (x^+, y^+) on the induced region such that

$$c_1 x^+ + d_1 y^+ = c_1 x^0 + d_1 y^0$$

(see Figure 1). The corresponding primal-dual nonconvex feasibility program is

$$\begin{aligned} c_1 x^+ + d_1 y^+ &= c_1 x^0 + d_1 y^0 \\ Ax^+ + By^+ &\le b \\ x^+, y^+, \lambda^+ &\ge 0 \end{aligned} \tag{3}$$

$$\begin{aligned} \lambda^+ B &\geq d_2 \\ \lambda^+(b - Ax^+) &= d_2 y^+. \\ (x^+, y^+) &\neq (x^0, y^0) \end{aligned}$$

If one introduces the *gap function*

$$g(x,y) = \max_{z | Bz \leq b - Ax, z \geq 0} d_2(z - y),$$

one can rewrite (3) as

$$\begin{aligned} 0 = \text{global} \min_{x,y} \quad & g(x,y) \\ \text{subject to} \quad & c_1 x + d_1 y = c_1 x^0 + d_1 y^0 \\ & (x,y) \in P \\ & (x,y) \neq (x^0, y^0). \end{aligned} \tag{4}$$

Notice that, unless (x^0, y^0) is already globally optimal for BLP, there exists at least one solution to the system (3). The generic procedure is described below.

ALGORITHM TABU

Step 1: (moving away from the current solution (x^0, y^0))

Through a pivot sequence, generate a point (x', y') that is

(i) "far" from (x^0, y^0), and
(ii) achieves a "high" value of the gap function g.

Simultaneously record the relevant Tabu information.

Step 2: (searching for a point in the induced region)

Starting from (x', y'), try to solve the optimization problem (4) by performing a sequence of Tabu moves.

□

Note that the Tabu algorithm, being a heuristic procedure, could fail to find a solution to (4), even if such a solution exists.

At Step 1 of algorithm TABU, we first move to a randomly selected adjacent vertex $(\hat{x}, \hat{y})$ of (x^0, y^0) in the polyhedron

$$P(x^0, y^0) = \{(x,y) \in P | c_1 x + d_1 y = c_1 x^0 + d_1 y^0\}.$$

Next we maximize the squared distance

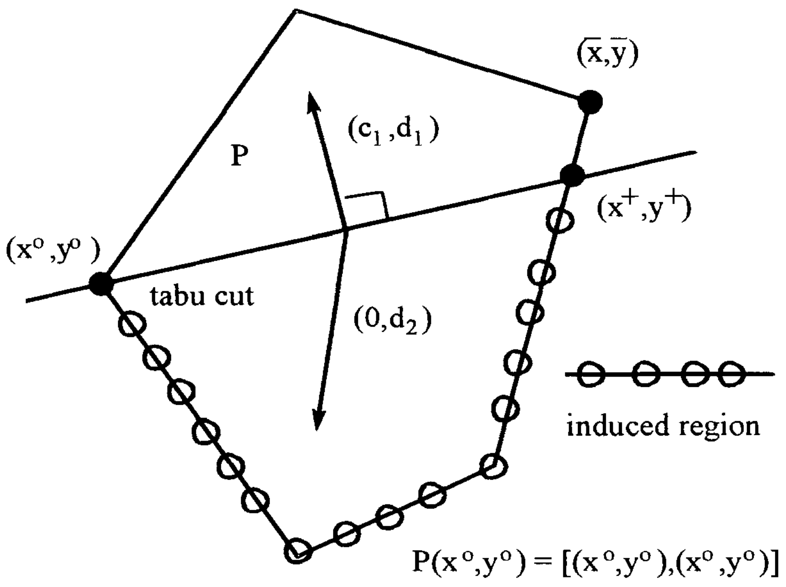

Figure 1. The Tabu cut.

$$\|(x,y) - (x^0,y^0)\|^2 \tag{5}$$

by performing one simplex step (pivot) for the linear program resulting from the linearization of the convex function (5) at the point $(\hat{x},\hat{y})$ (Frank-Wolfe direction-finding linear program at $(\hat{x},\hat{y})$). This process is repeated until a local maximum of (5) is reached. Since the distance function (5) is convex, this process generates a finite sequence of vertices of $P(x^0,y^0)$.

The Tabu methodology comes into play at Step 2 of the algorithm. Three types of Tabu tags are maintained throughout this phase: Tabus on entering variables, Tabus on exiting variables and Tabus on pivots, i.e. pairs of entering and exiting variables. The first two sets of tags prevent the reversal of past moves, while the third inhibits pivot repetitions. These Tabus remain active for a number of iterations generated randomly and uniformly within the intervals $[\underline{\theta}_{\text{in}},\overline{\theta}_{\text{in}}]$, $[\underline{\theta}_{\text{out}},\overline{\theta}_{\text{out}}]$ and $[\underline{\theta}_{\text{piv}},\overline{\theta}_{\text{piv}}]$, respectively.

Each iteration of the Tabu phase involves a move from a vertex of $P(x^0,y^0)$ to an adjacent vertex by means of a pivot operation. Only a subset of possible moves is considered at each iteration: the candidate list is made up of the k_1 most promising nonbasic variables obtained at the previous iteration, and is completed by selecting k_2 variables according to a cyclic management scheme. Hence, at each iteration, the length of the candidate list is at most $k = k_1 + k_2$. For each of these variables, we compute a "merit score" defined as the sum of two terms: the gap value that would result if the pivot were actually implemented and a penalty factor related to the Tabu status of the corresponding move. For a pivot involving r as the entering variable and s as the exiting variable, the penalty is given by the formula

$$\Pi(r,s) = \alpha([t_{\text{in}}(r) - t]^+ + [t_{\text{out}}(s) - t]^+ + 2[t_{\text{piv}}(r,s) - t]^+),$$

where α is a penalty weight factor, whose value decreases from one to zero as the number of Tabu iterations already performed increases, t is the current iteration index, $t_{\text{in}}(r)$ the instant (iteration index) at which the variable r is to be removed from the entering variables Tabu list, $t_{\text{out}}(s)$ the instant at which the variable s is to be removed from the exiting variables Tabu list, $t_{\text{piv}}(r,s)$ the instant at which the pair (r,s) is removed from the pivot Tabu list, and $[\,\cdot\,]^+$ denotes the maximum function:

$$[u]^+ = \max(0,u).$$

Note that, at the beginning of the Tabu step, the instants $t_{\text{in}}(r)$, $t_{\text{out}}(s)$ and $t_{\text{piv}}(r,s)$ are set to zero for all indices r and s.

While scanning the list of candidate pivots, the first (r,s) pivot that satisfies either of the following conditions is implemented:

1. (r,s) is not Tabu and decreases the gap.

2. (r,s) is Tabu and significantly decreases the gap (aspiration criterion). More precisely, (r,s) is implemented if the gap resulting from the (r,s) pivot is less than
$$\frac{1}{2} \times \text{current gap} \times \left(1 - \frac{\text{maxiter} - t}{\text{maxiter}}\right),$$
where "maxiter" is the maximum number of pivots allowed in the Tabu phase and t the current iteration index.

According to standard Tabu Search terminology, this second condition is called an "aspiration criterion", in the sense that the Tabu status of a pivot can be overriden if this allows the search to reach a particularly promising solution. If no pivot meets the previous two requirements, then we select the pivot that minimizes the previously defined merit score.

If, during the Tabu phase, the starting point (x^0, y^0) is rediscovered, the parameters $[\underline{\theta}_{\text{in}}, \overline{\theta}_{\text{in}}]$, $[\underline{\theta}_{\text{out}}, \overline{\theta}_{\text{out}}]$ and $[\underline{\theta}_{\text{piv}}, \overline{\theta}_{\text{piv}}]$ are increased, and the process restarted. In a symmetric fashion, if the gap value increases on two consecutive iterations, these parameters are decreased.

Once the Tabu phase is successfully completed, we try to improve on (x^+, y^+), using the heuristic algorithm TINI, which is nothing else than a reverse implementation of algorithm INIT, and is described below.

If the Tabu phase fails to discover a solution to (4) within "maxiter" pivots, the overall procedure halts.

ALGORITHM TINI

```
Step 0: Select a parameter $\Delta M$.
        Set $M$ to some suitably large value.
        Let $\lambda(M)$ be an optimal dual vector corresponding
        to the primal lower level vector $(x^+, y^+)$.

Step 1: Let $(x(M), y(M)) \in \arg\max_{(x,y)\in P} \{c_1x + d_1y + M[d_2y - \lambda(M)(b - Ax)]\}$

Step 2: Let $\lambda(M) \in \arg\min_{\lambda\in D} \{\lambda(b - Ax(M))\}$.

Step 3: if $d_2y(M) - \lambda(M)(b - Ax(M)) \neq 0$ then go to Step 4
            else set $M = M - \Delta M$ and GOTO Step 1.

Step 4: Output the feasible solution $(x(M + \Delta M), y(M + \Delta M))$.
```

□

At Step 0 of algorithm TINI, "suitably large" means that M is an exact penalty parameter or, in other words, any solution of the program (2) achieves a null gap value. Like algorithm INIT, algorithm TINI is completed by a local ascent procedure.

The overall procedure is illustrated in Figure 2.

3. Numerical results

The algorithm has been programmed in the FORTRAN language on a workstation HP730 operating under UNIX, and using Marsten's XMP linear programming code [14] for solving the linear subproblems encountered throughout. On the smaller problems, we could compare our algorithm against the exact algorithm of Hansen, Jaumard and Savard [10] programmed within the same computing environment. The exact algorithm of Hansen, Jaumard and Savard is currently one (if not "the") most efficient method for solving unstructured BLPs.

Series of test problems with sizes ranging from 40 to 200 variables and 20 to 200 constraints, with both medium sparse and dense matrix structures, have been solved. The problems were generated using a modification of Bard and Moore's generator [4] adopted in [10].

The results of the experiments are presented in Tables 1 to 6. In Tables 1 to 4 the figures are averaged over groups of ten problem instances.

The meaning of the column headings is given below:

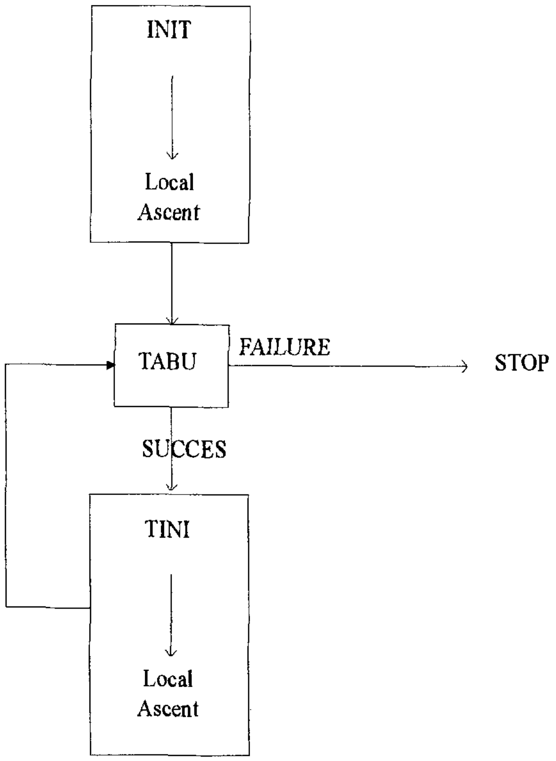

Figure 2. The hybrid Tabu-ascent algorithm

Table 1. Influence of the parameter "maxiter"

Problem parameters
$n_x = 50$ $n_y = 55$
$m = 42$ DENS=1.0
$\underline{\theta} = 11, \overline{\theta} = 15$
$k_1 = 10$ $k = 55$

maxiter	NSUC	NOPT	WORST	AVERAGE	CPU1	CPU2/CPU1
55	3	6	97.59	99.37	12.82	22.88
110	6	7	98.16	99.71	28.70	10.22
220	6	9	98.00	99.80	64.16	4.57

HTA:	hybrid tabu-ascent algorithm
EXACT:	exact algorithm HJS
n_x:	number of upper level variables
n_y:	number of lower level variables
m:	number of constraints
DENS:	density of the constraint matrix $[A \mid B]$
$\underline{\theta}$:	common value of the parameters $\underline{\theta}_{\text{in}}$, $\underline{\theta}_{\text{out}}$, $\underline{\theta}_{\text{piv}}$
$\overline{\theta}$:	common value of the parameters $\overline{\theta}_{\text{in}}$, $\overline{\theta}_{\text{out}}$, $\overline{\theta}_{\text{piv}}$
maxiter:	maximum number of iterations allowed in each Tabu phase
NSUC:	number of successful Tabu phases
NOPT:	number of problems solved to optimality
WORST:	worst heuristic-to-optimal ratio
AVERAGE:	average heuristic-to-optimal ratio
CPU0:	CPU time associated with algorithm INIT (including local search phase)
CPU1:	CPU time associated with the hybrid Tabu-ascent algorithm
CPU2:	CPU time associated with the exact algorithm HJS
k:	length of the list of candidate variables, sorted with respect to their respective merit scores
k_1:	number of most promising variables kept from the previous iteration in Tabu phase
OBJ:	value of first level objective
OBJ0:	value of first level objective achieved by algorithm INIT (including local ascent phase)
HEUR:	value of first level objective computed by HJS's initialization procedure
UPPER:	initial upper bound on the first level objective.

The numerical experiments were conducted in two stages. We first performed a series of tests to determine good values for the various parameters involved in the

Table 2. Influence of the parameter k

Problem parameters
$n_x = 50$ $n_y = 55$
$m = 42$ DENS=1.0
$\underline{\theta} = 11$, $\overline{\theta} = 15$
$k_1 = 10$ maxiter=110

k	NSUC	NOPT	WORST	AVERAGE	CPU1	CPU2/CPU1
27	3	5	97.59	99.20	16.85	17.41
55	6	7	98.16	99.71	28.70	10.22
105	4	6	97.84	99.43	39.99	7.33

Tabu phase of the algorithm. Problems with 50 upper level variables, 55 lower level variables, 42 constraints and density of 100% were used for these tests. These are difficult problems that are encountered in some practical situations; for instance, the BLP formulation of the nonparametric discriminant analysis problem leads to 100% dense constraint matrices (see Marcotte and Savard [16]).

In algorithm INIT, we observed that the penalty became exact for values of M ranging from 2 to 45. In algorithm TINI, the parameter M was initially set to 5, and increased by increments of .5 until the penalty became exact. In both algorithms, the value of ΔM was set to .5.

In Table 1 we assess the influence of the parameter "maxiter" on the performance of the algorithm. As expected, the quality of the solution improves as "maxiter" and CPU1 increase. However, it can be observed that increasing "maxiter" above the value 110 (twice the number of second level variables) led to minute improvements in the solution, while considerably increasing the computing time. In subsequent tests, "maxiter" has been set equal to twice the number n_y of lower level variables.

Some tests have also been conducted to determine a strategy for restricting the number of pivots to be considered. We introduced two parameters: (i) k governs the number of variables kept from the previous Tabu iteration, sorted with respect to their merit scores; (ii) k_1 controls the number of variables from the previous list of k variables to be evaluated in priority as entering variables. For each of these two parameters, 3 different values were considered. The results are shown in Tables 2 and 3. Based on these results, the values $k = n_y$ and $k_1 = 10$ have been retained. Note that the algorithm is quite insensitive to the value of the parameter k_1. Also, it proved unproductive to use the full list of variables; indeed, as k was increased from 55 to 105, the computing time increases *and* the solution deteriorates.

We also conducted a sensitivity analysis on the length of the Tabu list. The results are quite insensitive to this parameter. This behavior may be explained by the self-adjustment of this parameter and the use of an aspiration criterion.

Table 3. Influence of the parameter k_1

Problem parameters
$n_x = 50$ $n_y = 55$
$m = 42$ DENS=1.0
$\underline{\theta} = 11, \overline{\theta} = 15$
$k = 55$ maxiter=110

k_1	NSUC	NOPT	WORST	AVERAGE	CPU1	CPU2/CPU1
0	5	7	98.15	99.71	31.90	9.20
10	6	7	98.16	99.71	28.70	10.22
20	4	6	97.59	99.37	26.39	11.12

The main results are presented in Table 4, where we observe that the computing time CPU1 of our algorithm increases superlinearly with respect to the size of the problem, albeit at a much slower rate than the computing time CPU2 of the exact procedure HJS. With the exception of the smaller problems, CPU1 is much smaller than CPU2. An exact solution was obtained by the hybrid Tabu-ascent algorithm in 184 out of 250 problems, corresponding to a success rate of 73.6%. Even more impressive in our opinion is the fact that the relative error was, on the average, inferior to .5% and that it was very seldom superior to 5%. All these results were achieved in computing times 6 to 20 times faster than those of the exact algorithm on all but the smallest test problems. In several instances, the initialization algorithm INIT produced an optimal solution. In the next-to-last series of test problems, the performance of our algorithm is actually superior (!) to that of the exact algorithm. This is due to one extremely nasty instance of a 70×70 problem, which had to be halted when CPU2 exceeded the time limit value set to 15 000 seconds.

Table 5 contains a detailed account of the series corresponding to the parameters $n_x = 60$, $n_y = 65$, $m = 50$ and DENS= .4. The aggregated results for this series have previously been presented at line −4 of table 4. In five instances, the best solution was achieved at the end of the initialization phase, and two of these solutions are optimal. In the other five problems, where the Tabu phase actually improved upon the initial solution, the optimal solution was reached. In the two cases where the exact algorithm's computing time CPU2 was inferior to CPU1, algorithm INIT produced either an optimal or a near optimal (within .03% from optimality) solution. The computing times of algorithm INIT are reported in the 3rd column of Table 5. These figures are very low, except for the 4th instance, where the lower level problem was highly degenerate. Finally we notice that the coefficient of variation of the computing time CPU1 is very small, confirming the stability of the algorithm.

Table 4. Main results.

Problem parameters
$\underline{\theta} = \lceil n_y/5 \rceil$ $\overline{\theta} = \underline{\theta} + 4$
$k_1 = 10$ $k = n_y$ maxiter=$2n_y$

n_x	n_y	m	DENS	maxiter	NSUC	NOPT	WORST	AVERAGE	CPU1	CPU2
30	30	24	0.4	60	3	7	93.67	99.23	3.77	7.19
30	30	24	0.7	60	6	8	85.14	98.35	3.26	7.94
30	30	24	1.0	60	4	9	99.50	99.95	3.15	6.94
30	35	26	0.4	70	3	8	89.61	98.91	4.45	11.65
30	35	26	0.7	70	2	7	78.58	97.75	4.40	29.36
30	35	26	1.0	70	6	10	100.00	100.00	5.88	31.77
40	40	32	0.4	80	4	7	95.19	99.17	7.68	29.34
40	40	32	0.7	80	5	8	97.41	99.66	8.31	109.12
40	40	32	1.0	80	3	8	96.86	99.48	9.16	80.08
40	45	34	0.4	90	6	6	93.29	99.31	15.45	87.40
40	45	34	0.7	90	4	8	97.11	99.66	13.24	64.42
40	45	34	1.0	90	4	5	89.08	97.41	11.23	97.65
50	50	40	0.4	100	7	6	98.70	99.75	24.16	167.77
50	50	40	0.7	100	4	7	91.50	98.36	22.71	355.31
50	50	40	1.0	100	2	8	98.56	99.77	16.94	91.52
50	55	42	0.4	110	13	7	98.68	99.74	45.91	330.48
50	55	42	0.7	110	2	9	99.96	99.99	24.55	199.76
50	55	42	1.0	110	5	7	98.16	99.71	28.70	293.34
60	60	48	0.4	120	4	8	97.60	99.74	41.89	508.87
60	60	48	0.7	120	6	7	96.49	99.53	49.63	1680.99
60	60	48	1.0	120	3	9	96.89	99.69	35.97	585.62
60	65	50	0.4	130	6	7	98.01	99.64	69.86	1699.93
60	65	50	0.7	130	2	7	93.83	99.17	100.61	659.36
70	70	56	0.4	140	6	7	99.12	100.29*	88.56	1728.66
80	70	60	0.4	140	3	5	94.35	99.00	95.80	1163.10

Table 5. Detailed results.

Problem parameters
$n_x = 60$ $n_y = 65$
$m = 50$ DENS=0.4
$\underline{\theta} = 13$ $\overline{\theta} = 17$
$k_1 = 10$ $k = 65$ maxiter=130

			HTA				EXACT		
#	OBJ0	CPU0	NSUC	OBJ	CPU1	HEUR	UPPER	OBJ	CPU2
1	60.63	10.11	0	60.63	62.93	33.11	61.62	60.63	338.63
2	75.85	3.43	0	75.85	67.76	37.05	82.12	76.17	1257.61
3	75.16	5.20	2	76.38	96.97	39.96	81.61	76.38	793.65
4	64.17	80.08	1	65.28	142.84	33.65	68.10	65.28	579.28
5	79.00	7.64	1	79.93	54.95	63.25	82.14	79.93	551.09
6	79.47	1.79	0	79.47	31.07	43.05	79.84	79.47	6.00
7	62.24	3.53	1	62.26	81.79	44.71	62.38	62.26	47.27
8	84.18	8.95	0	84.18	52.75	34.82	91.15	85.20	11350.11
9	71.53	5.96	1	71.97	57.29	49.69	75.01	71.97	115.76
10	85.18	1.39	0	85.18	50.28	63.14	90.88	86.92	1959.94

Our last table (Table 6) contains the results of experiments performed on 10 large problems, for which a time limit of one hour computing time was imposed on the exact algorithm HJS. In all but one instance, the hybrid Tabu-ascent procedure generated the best solutions, in less than 13 minutes. In the other instance (9th problem), the solution values where identical, and the running times very similar. In five instances, the best solution was obtained at the end of the initialization phase, thus confirming the exceptional behavior of algorithm INIT completed by the local search phase.

4. Conclusions

In this paper we presented a hybrid Tabu Search-ascent heuristic for the approximate solution of large linear bilevel programs. This method performed very well on a large set of difficult test problems involving up to 200 variables and 200 constraints. Optimal solutions were identified for the vast majority of these, and relative errors were on the average extremely small. The computing times of the algorithm proved to be low and stable, from one problem to the other.

An integral part of our algorithm is the primal-dual startup procedure INIT, which performed beyond our expectations. Actually, INIT produced optimal or near-optimal solutions in many cases. On the other hand, the Tabu phase of the algorithm was successful in achieving its stated aim, that is, improving upon nonop-

Table 6. Results on large problems.

Problem parameters
$n_x = 90$ $n_y = 100$
$m = 76$ DENS=0.25
$\underline{\theta} = 20$ $\overline{\theta} = 24$
$k_1 = 10$ $k = 100$ maxiter=200

	HDT					EXACT		
#	OBJ0	NSUC	OBJ	CPU1	HEUR	UPPER	OBJ	CPU2
1	78.72	1	78.92	453.32	60.01	81.41	77.73	3600.00
2	118.02	0	118.02	274.72	89.28	118.37	116.67	3600.00
3	98.36	0	98.36	200.36	46.24	110.23	95.91	3600.00
4	80.96	0	80.96	331.11	53.00	89.91	77.87	3600.00
5	71.37	1	71.42	358.74	59.04	72.70	71.42	3600.00
6	92.85	1	96.90	752.22	68.03	100.87	93.17	3600.00
7	69.14	1	70.15	604.44	31.68	73.07	69.54	3600.00
8	92.54	0	92.54	183.21	60.41	94.40	80.50	3600.00
9	92.99	0	92.99	257.59	83.62	93.59	92.99	228.21
10	120.01	1	122.55	679.77	82.58	124.77	107.70	3600.00

timal initial solutions. In most cases where the startup procedure did not find an optimal solution, the Tabu phase did.

Our results clearly demonstrate that hybrid methods combining ascent and Tabu procedures can efficiently address difficult nonconvex or combinatorial optimization problems such as the linear bilevel programming problem. It now remains to assess the capability of our approach to tackle other classes of problems with similar structures. In particular, we want to consider bilinear bilevel programs of the form

$$\begin{array}{ll} \max\limits_{T} & Tx \\ \min\limits_{x,y} & (c+T)x + dy \\ & Ax + By \geq b, \end{array}$$

which arise in the context of optimal pricing of commodities.

Note

1. Throughout the paper, left vectors (cost coefficients and dual vectors) are assumed to be row vectors while right vectors (resource coefficients and primal vectors) are assumed to be column vectors, thus eliminating the need for the transposition symbol.

References

1. Anandalingam, G. and White, D.J., *A solution method for the linear static Stackelberg problem using penalty functions*, IEEE Transactions on Automatic Control, 35 (1990), pp. 1170–1173.
2. Bard, J.F. and Falk, J.E., *An explicit solution to the multi-level programming problem*, Computers and Operations Research, 9 (1982), pp. 77–100.
3. Bard, J. F., *An efficient point algorithm for linear two-stage optimization problem*, Operations Research, 31 (1983), pp. 670–684.
4. Bard, J. F. and Moore, J.T., *A branch and bound algorithm for the bilevel programming problem*, SIAM Journal on Scientific and Statistical Computing, 11(2) (1990), pp. 281–292.
5. Bialas, W.F. and Karwan, M.H., *On two-level linear optimization*, IEEE Transactions on Automatic Control, AC-27(1) (1982), pp. 211–214.
6. Candler, W. and Townsley, R., *A linear two-level programming problem*, Computers and Operations Research, 9 (1982), 59–76.
7. Glover, F., *Tabu Search, Part I*, ORSA Journal on Computing 1 (1990), pp. 190–206.
8. Glover, F., *Tabu Search, Part II*, ORSA Journal on Computing 2 (1990), pp. 4–32.
9. Hansen, P., *The steepest ascent mildest descent heuristic for combinatorial programming*, Congress on Numerical Methods in Combinatorial Optimization, Capri, 1986.
10. Hansen, P., Jaumard, B. and Savard, G., *New branch-and-bound rules for linear bilevel programming*, SIAM Journal on Scientific and Statistical Computing 13 (1992), pp. 1194–1217.
11. Haurie, A., Savard, G. and White, D. J., *A note on: An efficient point algorithm for a linear two-stage optimization problem*, Operations Research, 38 (1990), pp. 553–555.
12. Jeroslow, R.G., *The polynomial hierarchy and a simple model for competitive analysis*, Mathematical Programming, 32 (1985), pp. 146–164.
13. Júdice, J. and Faustino, A., *A sequential LCP method for bilevel linear programming*, Annals of Operations Research, 34 (1992), pp. 89–106.
14. Marsten, R.E., *The design of the XMP linear programming library*, Transactions on Mathematical Software, 7(4) (1981), pp. 481–497.
15. Mathieu, R., Pittard, L. and Anandalingam, G., *Genetic algorithm based approach to bi-level linear programming*, R.A.I.R.O. Recherche Opérationnelle, 28 (1994), pp. 1–21
16. Marcotte, P. and Savard, G., *Novel approaches to the discrimination problem*, Zeitschrift für Operations Research, 36 (1992), pp. 517–545.
17. Savard, G. and Gauvin, J., *The steepest descent direction for the nonlinear bilevel programming Problem*, Operations Research Letters, 15 (1994), pp. 265–272.
18. Vicente, L.N. and Calamai, P.H., *Bilevel and multilevel programming: A bibliography review*, forthcoming in Journal of Global Optimization.
19. Vicente, L., Savard, G. and Júdice, J., *Descent Approaches for Quadratic Bilevel Programming*, Journal of Optimization Theory and Applications, 81 (1994), pp. 379–399.