ALGORITHMS FOR STATIC STACKELBERG GAMES WITH LINEAR COSTS AND POLYHEDRA CONSTRAINTS*

George P. Papavassilopoulos

Department of Electrical Engineering-Systems
University of Southern California
Los Angeles, California 90089

Abstract

Three algorithms for solving the static Stackelberg game with two decision makers, linear costs and polyhedra constraints are presented. The nonconvexity of the Reaction set of the Follower is a major source of difficulty in solving such problems. The algorithms presented are able to bypass this difficulty and find the solution in a finite number of iterations.

Introduction

The Stackelberg game was first stated by von Stackelberg in 1934, [1], in its static form and much later reintroduced in a dynamic framework by Cruz, Chen and Simaan in 1972, [2-4] and studied further in [5-9] and elsewhere. During the last years, a lot of research has been directed toward studying this problem because of its importance in hierarchical decision making, incentives [9], etc. Most of this work has concentrated on the dynamical and informational aspects and very little has been done on the algorithmic aspect even for simple static cases, like the one that is the object of this paper. The only existing papers in the literature — that the author is aware of — addressing directly this area are [10-16]. The works of [10,15,16] can be considered as addressing algorithms for particular Stackelberg games although no reference was made there to the general Stackelberg problem. In this paper we present three algorithms for solving the deterministic static Stackelberg game, with two levels of hierarchy, where the costs of both the Leader and the Follower are linear functions of their decisions and the vector of both the decisions has to lie in a polyhedron in a finite dimensional Euclidean space. A major source of difficulty for finding the solution is the nonconvex character of the Follower's reaction set. Although one could claim that in princple the global solution can be found by finding all the vertices of the constraint polyhedron, this is a highly impractical method to employ; the algorithms presented here find the global optimum without having to do that. The relative immaturity of the area of algorithms for Stackelber problems and the lack of sufficient computational experience restrain us from making — at the present time — definitive statements about the superiority of several algorithms over others. Thus, no relative evaluation of the algorithms presented here is attempted.

The structure of this paper is as follows. In Section 2 we state the problem and elaborate on the content of [10-16] where related work has been reported. In Sections 3, 4 and 5 we present three algorithms, provide their geometrical interpretation and discuss some of their particular characteristics, shortcomings, as well as ways to facilitate their implementation. In the last Section 6, we discuss differences among the algorithms, situations in which the one could be preferred over the other and delineate directions for further research.

*This work was supported in part by the U.S. Air Force Office of Scientific Research under Grants AFOSR -80-0171, and AFOSR-82-0174.

PROBLEM STATEMENT

The real vectors $c_1 \in R^{n_1}$, $c_2 \in R^{n_2}$, $b \in R^m$ and the real matrices $A_1(mxn_1)$, $A_2(mxn_2)$ are given. x_1 and x_2 denote vectors in R^{n_1} and R^{n_2} respectively. (R^k denotes the k-th dimensional Euclidean space and "'" denotes transposition).

For *fixed* x_1 we solve the problem

$$\begin{aligned} &\text{minimize } d'x_2 \\ &x_2 \\ &\text{subject to: } A_2x_2 \le b - A_1x_1 \end{aligned} \tag{1}$$

and we denote the solution set by $T(x_1)$. Subsequently we solve the problem

$$\begin{aligned} &\text{minimize } c_1'x_1 + c_2'x_2 \\ &x_1, x_2 \\ &\text{subject to } x_2 \in T(x_1) \end{aligned} \tag{2}$$

A pair (x_1^*, x_2^*) which solves (2) is called a *Stackelberg equilibrium*. Our aim is to state and analyze algorithms for finding such a pair. It is assumed throughout this paper, that the constraint set

$$\delta = \{(x_1,x_2): A_1x_1 + A_2x_2 \le b\} \tag{3}$$

is bounded and nonempty. This assumption guarrantees that $T(x_1) \neq \phi$ for some x_1 so that (2) is a nontrivial problem and does have a solution.

The problem stated above describes the following situation. There are two decision makers, the Leader -1-, who chooses x_1 and wants to minimize $J_1(x_1,x_2) = c_1'x_1 + c_2'x_2$, and the Follower -2-, who chooses x_2 and wants to minimize $J_2(x_2) = d'x_2$. x_1 and x_2 are subject to the joint constraint $(x_1,x_2) \in \delta$. The Leader chooses x_1 first, and for a given choice of x_1 the Follower solves (1). If we denote by $T(x_1)$ the solution of (1), then the incurring cost to the Leader is $J_1(x_1,T(x_1)) = c_1'x_1 + c_2'T(x_1)$ (assuming $T(x_1)$ has a single element denoted again $T(x_1)$, by abusing the notation). Thus, the Leader's problem is to minimize $c_1'x_1 + c_2'T(x_1)$ over x_1. (By convention, if $T(x_1) = \phi$ we set $J_1(x_1,x_2) = +\infty$.) In general, if (x_1^*,x_2^*) solves (2) x_2^* will be in $T(x_1^*)$, but $T(x_1^*)$ might be multivalued, so that if the Leader declares $x_1 = x_1^*$, the Follower might choose $\tilde{x}_2 \in T(x_1^*)$ $\tilde{x}_2 \neq x_2^*$, in which case $d'\tilde{x}_2 = d'x_2^*$, $c_1'x_1^* + c_2'\tilde{x}_2 \le c_1'x_1^* + c_2'x_2^*$. Thus, if the Follower wants to hurt the Leader he will choose — if possible — $\tilde{x}_2 \in T(x_1^*)$ with $c_1'x_1^* + c_2'\tilde{x}_2 < c_1'x_1^* + c_2'x_2^*$. Attributing such a desire to the Follower means that $J_2(x_2)$ does not really represent accurately the Follower's objective and we could have

CH1788-9/82/0000-0647$00.75 © 1982 IEEE

considered a new d equal to the old one minus εc_2, for some $\varepsilon > 0$. Nonetheless, even if the Follower does not intend to hurt the Leader, he could choose $\tilde{x}_2$ because of ignorance. If the Leader wants to safeguard himself against such a case, he would rather solve

$$\underset{x_1}{\text{minimize}} \quad \underset{x_2}{\text{maximize}} \; c_1'x_1 + c_2'x_2 \qquad (2')$$
$$\text{subject to } x_2 \in T(x_1)$$

instead of (2). The first two algorithms presented deal with the case where (2) is employed, whereas the third one can also handle the case where (2') is employed.

Although the problem (1),(2) is quite easy to pose its solution is not trivial. One of the main reasons for it is that the *reaction set* of the Follower, defined as

$$\mathcal{R} = \{(x_1,x_2), x_2 \in T(x_1)\} \qquad (4)$$

may not be convex. $\mathcal{R}$ is piecewise linear, lies on the boundary of $\mathcal{S}$ and its vertices are vertices of $\mathcal{S}$. (2) can now be written equivalently as

$$\underset{x_1,x_2}{\text{minimize}} \; c_1'x_1 + c_2'x_2 \qquad (5)$$
$$(x_1,x_2) \in \mathcal{R}$$

Although $\mathcal{R}$ might not be a convex set, (5) can be considered in principle as a Linear Programming problem, after substituting equivalently $\mathcal{R}$ in (5) by its convex hull. This idea is exploited by the third algorithm which is presented in Section 5. (Substitution of $\mathcal{R}$ in (5) by its convex hull will not introduce any new global minima located at vertices of $\mathcal{R}$ although it might introduce new global minima which lie on edges of the convex hull of $\mathcal{R}$ which are not in $\mathcal{R}$.) That the solution of (5) exists and is achieved at some vertex of $\mathcal{R}$ is obvious and thus by bringing our problem (1),(2) to the form (5) we conclude that it does have a solution which is achieved at some vertex of $\mathcal{S}$.

The only papers, that we are aware of, addressing similar problems are [10-16]. In [10], Falk considers (1)-(2) in the special case where $c = -c_2$ and shows that then (2) can be considered equivalently as a problem of minimizing a piecewise linear convex function over a polyhedron (Theorem 2.1 in [10]). The reason that this is true is that if $d = -c_2$ then the projection of $\mathcal{R}$ on the space spanned by d and x_1 has nice convexity properties; this is not true if $d \neq -c_2$. The contribution of [10] consists of an algorithms for solving the problem (1)-(2) with $d = -c_2$. In Theorem 2.5 of [10], Falk also proves that the decision maker's 1 incurring cost $(c_1'x_1 + c_2'x_2)$ is better if decision maker 1 is the Leader than his incurring cost if he were the Follower. The second algorithm presented in this paper can be considered as a generalization of Falk's algorithm. In [11] several algorithms were described for solving static Stackelberg games with nonlinear costs and constraints. The basic aim of [11] was to propose sensible algorithms and to provide intuitive justifications. One of the algorithms of [11] concerning the linear case, modified and extended appropriately as to guarantee convergence to a global optimum is the first algorithm presented here. An algorithm very similar to the one of [11] concerning the linear case, was almost concurrently reported in [12]. Both the schemes of [11] and [12] though, have the handicap that they might converge to local solutions. Two other algorithms were also presented in [13], but they seem to require an extensive amount of computation. In [14], an algorithm for solving static Stackelberg games with nonlinear costs and constraints was presented. The idea of this algorithm is to replace $x_2 \in T(x_1)$ by the corresponding Kuhn-Tucker conditions, view the Leader's problem as a constrained classical nonlinear programming problem and employ a barrier method for solving it. This algorithm yields a local optimum and no effort was made in [14] to tackle the problems due to the nonconvex character of the reaction set. Finally, the algorithms of [15] and [16] can be considered as addressing the special case of the Stackelberg problem where the constraints for x_1 and x_2 are decoupled, i.e., $x_1 \in X_1$, $X_2 \in X_2$, where X_1 and X_2 are (independent) subsets of R^{n1}, R^{n2}, respectively (In [15], X_2 is assumed compact). The decoupling of the constraints on x_1 and x_2 is not always realistic for the Stackelberg problem, since if for example we have a linear cost for the Follower, the Follower becomes independent of the Leader. For more explanations concerning the relation of the algorithms in [15], [16] with the Stackelberg problem, see [11].

FIRST ALGORITHM

The algorithm of this section is motivated by the following considerations. If the Leader declares $x_1 = x_{1k}$, then the Follower solves (1) with $x_1 = x_{1k}$, (fixed), and finds $x_2 = x_{2k}$. The Leader knowing (x_{1k},x_{2k}) wishes to declare a new $x_1 = x_{1,k+1}$, so that when the Follower solves (1) with $x_1 = x_{1,k+1}$, the resulting $x_{2,k+1}$ will be such that $c_1'x_{1,k+1} + c_1'x_{1,k+1} < c_1'x_{1k} + c_2'x_{2k}$. This means that the sequence $\{(x_{1k},x_{2k})\}$ should be in $\mathcal{R}$ and should be moving so that the Leader's cost will be decreasing. Caution should be exercised as to insure convergence to a global, and not a local, minimum. This is taken care in Step 3 by introducing appropriately new cuts (constraints). These considerations suggest the following algorithm.

Step 1: For given x_1, solve (1) and find x_2. Set $(x_{1k}, x_{2k}) = (x_1,x_2)$ and go to Step 2, with $k = 0$.

Step 2(k): When at (x_{1k},x_{2k}) examine the neighboring vertices of (x_{1k},x_{2k}) and find those which are in $\mathcal{R}$ and also strictly reduce the Leader's cost; if $(\bar{x}_{1k},\bar{x}_{2k})$ is any neighboring vertex of (x_{1k},x_{2k}) with these two properties, set $(x_{1,k+1},x_{2,k+1}) = (\bar{x}_{1k},\bar{x}_{2k})$ and repeat Step 2 for k+1. If there is no such neighboring vertex, go to Step 3(k).

Step 3(k): Consider the constraint

$$c_1'x_1 + c_2'x_2 \leq c_1'x_{1k} + c_2'x_{2k} = b_k \qquad (6)$$

and adjoin it to (3), so that the new constraint set is

$$S_k = \{(x_1,x_2): \; A_1x_1 + A_2x_2 \leq b, \; c_1'x_1 + c_2'x_2 \leq b_k\} \qquad (7)$$

Let $\mathcal{R}_k$ be the reaction set of the Follower subject to the constraint (7) (and not (3)). Set $(x_{1,k+1},x_{2,k+1})$ equal to any neighboring vertex of (x_{1k},x_{2k}) in S_k which is in $\mathcal{R}_k$ and go to Step 2(k+1) ($(x_{1,k+1},x_{2,k+1})$ satisfies (6) as an equality). If repeated use of Steps 2 and 3 with examination of all the candidate vertices fails to produce a decrease in the Leader's cost, stop and you have found a global optimum.

There are several points to be made concerning the operation of this algorithm.

i) To start the algorithm we need some (x_1,x_2) in $\mathcal{S}$ as to set $x_{10} = x_1$. This initial feasible point can be found the same way as in the classical linear programming case.

ii) The implementation of the algorithm can be done by bringing the constraints to standard form setting up the usual tableau forms as in the simplex method and removing and introducing vectors to the basis.

iii) Checking whether a neighboring vertex of (x_{1k},x_{2k}) is in $\mathcal{R}$ can be done by verifying whether for $x_1 = x_{1k}$, x_{2k} solves (1), i.e., if it satisfies the familiar Linear Programming stationarity conditions. At Step 2 we might have available several vertices neighboring $(x_{1k},$

x_{2k}) which are in $\mathcal{R}$ and strictly reduce the Leader's cost, although we do not need but only one; we might then choose the one which results to the largest decrease of the Leader's cost, or the one which lies on the edge as close as possible to $(-c_1,-c_2)$, (i.e., steepest descent).

iv) If when at Step 2, the case arises that we have to go to Step 3, this means that (x_{1k},x_{2k}) is a local minimum of (5). By going to Step 3 we intend to check whether (x_{1k},x_{2k}) is a global minimum or whether we can somehow jump to another point in $\mathcal{R}$ — using a path not necessarily in $\mathcal{R}$ — from which we can decrease further the Leader's cost by moving on $\mathcal{R}$.

v) When repeated use of Steps 2 and 3 results finally to a point which strictly reduces the Leader's cost, then the constraint (6) will be dropped.

vi) If repeated use of Steps 2 and 3 fails to produce a lower value for the Leader and this has happened after all possible edges have been employed in Step 3, this means that the global minimum of (2) has been found.

vii) When using Steps 2 and 3 we have to take precaution so that cycling will not occur; i.e., what can happen is that during several iterations the algorithm will not decrease the Leader's cost and will be returning to the same point. If the same point appears again we repeat Step 3, but we choose to go to a neighboring vertex different than the one we choose when we first used Step 3. Nonetheless, a vertex which appeared during the first cycle can appear *after* some iterations were performed when we started from a different neighboring vertex. Then again we start from another neighboring vertex and so on. Thus, we might end up with a situation when all possible vertices for which (6) is active and are in the reaction set of the Follower with constraint set (7) have to be checked. If none of them can yield a direction for which the Leader's cost decreases, then we are at a global optimum. This situation might result in having to check very many vertices.

viii) If we reach a point in $\mathcal{R}$ and it is such that we have to go to Step 3, then there are several directions that we can take which are on the same plane $c_1x_1 + c_2x_2 =$ constant. We might benefit by using first the direction closest to the last direction on $\mathcal{R}$ along which the Leader's cost was decreased.

ix) If $d = -c_2$, then there will be at most two cases that we will need to introduce a constraint of the form $c_1x_1 + c_2x_2 \leq$ some constant. This will happen if we have reached the global optimum and we will check its global optimality or when we will have achieved a local optimum. Notice, that in this case, even if there are many local (non-global) optima, the value of the Leader's cost is the same at all of them.

x) Several known techniques addressing issues due to degeneracy can be appropriately employed and the algorithm will effectively terminate in a finite number of iterations.

SECOND ALGORITHM

The algorithm of this section constitutes an extension of Falk's algorithm, see [10], where the case $c_2 = -d$ was considered. The algorithm is based on the following idea. First solve the problem

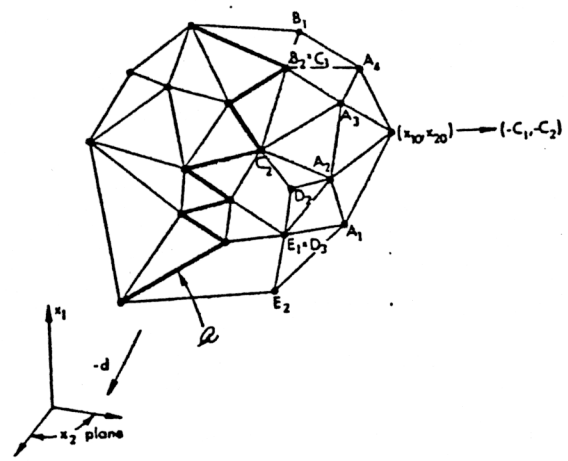
$$\min\ c_1'x_1 + c_2'x_2$$

$$x_1, x_2$$

$$\text{subject to: } A_1x_1 + A_2x_2 \leq b$$

The solution (x_{10},x_{20}) is achieved at some vertex of the polyhedron (3). If for $x_1 = x_{10}$, x_{20} solves (1), i.e., if $(x_{10},x_{20}) \in \mathcal{R}$, then we have solved the problem. If not, check whether any of the neighboring vertices of (x_{10},x_{20}) is in $\mathcal{R}$. Check whether any of the neighboring vertices of the first mentioned neighboring vertices of x_{10},x_{20} is in $\mathcal{R}$ and so on. In other words start branching away from x_{10},x_{20}, and try to find vertices which are in $\mathcal{R}$. The first time we find a vertex in $\mathcal{R}$ which incurs some cost J_1^1 to the Leader, we have a candidate as solution. The branching directions which result to vertices with bigger costs for the Leader are discarded, but we keep on the branching procedure for the rest till some of them yields another point of $\mathcal{R}$ with cost J_1^2 for the Leader smaller than J_1^1. We continue in this fashion with J_1^{K+1} assuming the role of J_1^K until at J_1^N we reach a point where all the new vertices result to costs bigger than J_1^N.

A geometric interpretation of Algorithm 2 is depicted in Figure 1. From (x_{10},x_{20}) we go to A_1,A_2,A_3,A_4. We check whether any of the A_1,A_2,A_3,A_4 is in $\mathcal{R}$ and the answer is negative. From A_4 we branch out to B_1,B_2 from A_3 to $C_1 = B_2,C_2$ from A_2 to $D_1 = C_2,D_2,D_3$ and from A_1 to $E_1 = D_3,E_2$. We check whether any of the B_1, B_2,C_2,D_2,D_3,E_2 is in $\mathcal{R}$ and the answer is yes for B_2 and C_2. We calculate the value of the Leader's cost at B_2, C_2, denote them by $J_1(B_2)$, $J_1(C_2)$ and set $J_1^1 = \min(J_1(B_2), J_1(C_2))$. Let $J_1^1 = J_1(B_2)$. If $J_1(E_2) > J_1^1$ we do not need to continue branching from E_2. If $J_1(D_3) < J_1^1$ we will continue branching from D_3 since we do not know yet whether a branching starting from D_3 will yield a cost for the Leader which is less than J_1^1. Continuing branching away from (x_{10},x_{20}) we reach a stage where all new branchings yield vertices which yield cost for the Leader greater than J_1^1, (independently of whether they are in $\mathcal{R}$ or not). From this moment on we know that even if we meet a point of $\mathcal{R}$, it will yield a cost for the Leader which is higher than J_1^1 and we conclude that B_2 is the solution.



These considerations suggest the following algorithm:

Step 1: Solve the problem

$$\min\ c_1'x_1 + c_2'x_2$$

$$\text{subject to: } A_1x_1 + A_2x_2 \leq b \qquad (8)$$

Let the solution be (x_1^0,x_2^0). If for $x_1 = x_1^0,x_2^0$ solves (1), then (x_1^0,x_2^0) is a solution. Otherwise go to Step 2.

Step 2: Find the neighboring vertices of (x_1^0,x_2^0) call them $Q_1,Q_2,\ldots,Q_k$, $Q_i = (x_1^i,x_2^i)$. Set $c_1x_1^i + c_2x_2^i = u_1^i$ and check whether for any $x_1 = x_1^i,x_2^i$ solves (1), i.e., check

whether $Q_i \in \mathcal{R}$. Let $i = 1,\ldots,\ell$ correspond to the Q_i's with $Q_i \in \mathcal{R}$. Set $u_1^* = \min\{c_1x_1^i + c_2x_2^i,\ Q_i \in \mathcal{R}\}$. If no Q_i is in $\mathcal{R}$, $u_1 = +\infty$, go to Step 3.

Step 3: Find the neighboring vertices of each Q_i that appeared in Step 2, for which $Q_i \notin \mathcal{R}$ (i.e., $i = \ell+1,\ldots,k$). Let $Q_1^i, Q_2^i, \ldots Q_m^i$ be the neighboring vertices of each such Q_i. Retain only those Q_j^i's for which the value of J_1 is $\leq u_1^*$. For these Q_j^i's, check which ones are in $\mathcal{R}$. Set $u_2^* = \min\{u_1^*, [\min_{i,j}(J_1(Q_j^i), Q_j^i \in \mathcal{R})]\}$. Go to Step 4.

Step 4: Find and retain all the adjacent vertices of the Q_j^i's which were retained at Step 3, which are in $\mathcal{R}$ and at which the value of J_1 is $\leq u_2^*$. Go to Step 5.

$\vdots$

Step k: Find and retain all the adjacent vertices of the vertices retained at Step k-1, which are in $\mathcal{R}$ and for which the value of J_1 is $\leq$ last u_ℓ^*. If for some of them the value of J_1 is strictly less than u_ℓ^*, go to Step k+1; if not, stop and you have found a solution.

There are several points to be made concerning the operation of this algorithm.

i) A certain vertex might appear as a neighbor of two vertices or reappear at a certain stage. This should cause no confusion since we can provide in the algorithm for not checking the same vertes twice.

ii) Since $u_1^* \geq u_2^* \geq \ldots \geq u_\ell^*$ the value u_ℓ^* with which we compare the values of J_1 at several vertices decreases, i.e., each step we restrict ourselves to vertices which result in a cost for the Leader less that any cost for him that has occurred before k at vertices lying in $\mathcal{R}$ that we have already met. Notice that if u^* is the optimum value of J_1 in (2), it obviously holds $u_1^* \geq \ldots \geq u_k^* \geq \ldots \geq u^*$.

iii) If some vertex is found to be in $\mathcal{R}$ we can stop branching away from it since this branching will either give a bigger cost for the Leader or will yield vertices that can be reached by branching away from other vertices which are not in $\mathcal{R}$. Of course we can branch away from such vertices if we so wish, as in a case where we know that such a vertex is very close to the solution.

iv) Checking whether a vertex is in $\mathcal{R}$ can be done by using the stationarity conditions of linear programming. The following considerations can also facilitate this checking. If x_2^* solves the problem minimize $d\,x_2$ subject to $A_2x_2 \leq b - A_1x_1$, (x_1 = fixed), then $d_1 + A_2'\mu = 0$ for some $\mu = (\mu_1,\ldots,\mu_m)$, $\mu_i \geq 0$. Activity of some constraint is obviously implied by having that the corresponding μ_i is $\neq 0$. To that effect we state the following Proposition which generalizes Theorem 3.1 of [10] where only part (i) is given.

Proposition: Let $A_2' = [a_{21}, a_{22}, \ldots, a_{2m}]$ and consider the relationship

$$d + \mu_1 a_{21} + \mu_2 a_{22} + \ldots + \mu_m a_{2m} = 0,\ \mu_1, \mu_2, \ldots, \mu_m \geq 0.$$

$c, a_{ij} \in R^{n2}$, $\mu_i \in R$. Assume that none of the vectors $d, a_{21}, \ldots, a_{2m}$ is zero. Let

$$(\gamma_{ij}) = \begin{bmatrix} \gamma_{00} & \gamma_{01} & \ldots & \gamma_{0m} \\ \gamma_{10} & \gamma_{11} & \ldots & \gamma_{1m} \\ \vdots & \vdots & & \\ \gamma_{m0} & \gamma_{m1} & \ldots & \gamma_{mm} \end{bmatrix} = \begin{bmatrix} d' \\ \hline A_2' \end{bmatrix} \quad [d;\ A_2']$$

Then:

i) At least one μ_i with $i \in \{i:\gamma_{0i} < 0,\ i = 1,2,\ldots,m\}$ is > 0.

ii) If $\gamma_{i0} \geq 0$, then at least one of the μ_j's with $j \in \{j:\gamma_{ij} < 0,\ j = 1,\ldots,m\}$ is > 0.

iii) If $\gamma_{i0} \leq 0$, then at least one of the μ_j's with $J \in \{j:\gamma_{ij} > 0,\ j = 1,\ldots,m\}$ is > 0.

iv) If $\gamma_{00}, \gamma_{01}, \ldots, \gamma_{0m}$ are all ≥ 0, then the μ_i's which correspond to $\gamma_{0i} > 0$ are zero.

v) If $\gamma_{i0}, \gamma_{i1}, \ldots, \gamma_{im}$ are all ≥ 0, then the μ_j's which correspond to $\gamma_{ij} > 0$ are zero.

Proof: The proof is easy and can be obtained by inspecting the relationships:

$$\gamma_{00} + \mu_1\gamma_{01} + \ldots + \mu_m\gamma_{0m} = 0$$

$$\gamma_{i0} + \mu_1\gamma_{i1} + \ldots + \mu_m\gamma_{im} = 0 \qquad \bullet$$

and using the nonnegativity of the μ_i's. □

By using this Proposition we can make some conclusions about the activity of certain constraints.

v) When we examine a vertex $Q = (\bar{x}_1, \bar{x}_2)$ at some stage k, and it turns out that $Q \in \mathcal{R}$, we can still go ahead and solve (1) with $x_1 = \bar{x}_1$. If the solution is $\tilde{x}_2$, then we can take u_{k-2}^* as the minimum not only of the quantities mentioned explicitly in Step k but also of $c_1\bar{x}_1 + c_2\tilde{x}_2$. Thus we will have an even smaller u^*, thing which facilitates the exclusion of several vertices.

vi) The implementation of generating the neighboring vertices can be done by bringing the constraints into standard form and by changing the basis. Also the case of degeneracy needs special care for reasons essentially the same as those of the classical Linear Programming problem.

vii) This algorithm will obviously converge in a finite number of iterations after having generated and checked all the vertices at which the value of J_1 is greater or equal than u^*.

THIRD ALGORITHM

The basic idea of this algorithm emanates from the fact that the Leader's problem can be actually written down equivalently as a Linear Programming problem. This is due to the fact that minimizing $c_1'x_1 + c_2x_2$ subject to $(x_1, x_2) \in \mathcal{R}$ is equivalent to minimizing $c_1'x_1 + c_2x_2$ subject to $(x_1, x_2) \in$ convex hull of $\mathcal{R}$. This is so because the extreme points of $\mathcal{R}$ are the same with those of its convex hull and the solution of (2) can anyway be achieved at an extreme point of $\mathcal{R}$. In the following we construct a Linear Programming problem equivalent to (2). Using the Kuhn-Tucker-Karush Conditions we obtain that (2) is equivalent to:

$$\left.\begin{array}{ll} \text{minimize } c_1'x_1 + c_2'x_2 & \\ x_1, x_2, \mu & \\ \text{subject to: } d + A_2'\mu = 0 & \\ A_1x_1 + A_2x_2 \leq b & \\ \mu'(A_1x_1 + A_2x_2 - b) = 0 & \\ \mu \geq 0 & \end{array}\right\} \qquad (9)$$

Let

$$A_1 = \begin{bmatrix} a_{11}' \\ \vdots \\ a_{1m}' \end{bmatrix},\ A_2 = \begin{bmatrix} a_{21}' \\ \vdots \\ a_{2m}' \end{bmatrix},\ b = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix},\ \mu = \begin{bmatrix} \mu_1 \\ \vdots \\ \mu_m \end{bmatrix}$$

$$I = \{1,2,\ldots,m\}$$

$$I_1^j = \text{any subset of } I,\ j = 1,w,\ldots,N,\ (N = 2^m)$$

$$I_2^j = I - I_1^j$$

The constraint set of (9), can be written equivalently as

$$\left.\begin{aligned} & A_1x_1 + A_2x_2 \le b \\ & d + \mu_1^j a_{21} + \ldots + \mu_m^j a_{2m} = 0 \\ & a'_{1i} + a'_{2i}x_2 = b_i,\ i \in I_1^j \\ & a'_{1i}x_1 + a_{2i}x_2 \le b_i,\ i \in I_2^j \\ & \mu_i^j \ge 0,\ i \in I_1^j \\ & \mu_i^j = 0,\ i \in I_2^j \\ & \mu^j = (\mu_1^j, \mu_2^j, \ldots, \mu_m^j)' \end{aligned}\right\} \quad (C'_j)$$

for some $j \in \{1,2,\ldots,N\}$

Notice that some (C'_j) might be empty. If a pair (x_1,x_2) let us call it (x_1^j, x_2^j), is such that for some μ^j (C'_j) holds, this means that (x_1^j, x_2^j) is in $\mathcal{X}$. Any admissible (x_1,x_2) in (9) can be written as

$$\left.\begin{aligned} x_1 &= \lambda_1 x_1^1 + \ldots + \lambda_N x_1^N \\ x_2 &= \lambda_1 x_2^1 + \ldots + \lambda_N x_2^N \end{aligned}\right\} \begin{array}{l} (x_1^j, x_2^j) \text{ satisfy } (C'_j) \text{ for some } \mu^j. \\ j = 1,\ldots,N \end{array}$$

$$\lambda_1,\ldots,\lambda_N \ge 0,\ \lambda_1 + \ldots + \lambda_N = 1$$

Notice that we can multiply all the relationships in (C'_j) by λ_j and as long as (C'_j) is empty, the corresponding λ_j will be 0. Thus, using $\bar{x}_i^j = \lambda_j x_i^j$, $\bar{\mu}_2^j = \lambda_j \mu_2^j$ we transform (C'_j) to (C''_j)

$$\begin{aligned} & \lambda_j d + \bar{\mu}_1^j a_{21} + \ldots + \bar{\mu}_m^j a_{2m} = 0 \\ & a'_{1i}\bar{x}_1^j + a'_{2i}\bar{x}_2^j = \lambda_j b_i \quad i \in I_1^j \\ & a'_{1i}\bar{x}_1^j + a_{2i}\bar{x}_2^j \le \lambda_j b_i \quad i \in I_2^j \qquad (C''_j) \\ & \bar{\mu}_i^j > 0,\ i \in I_1^j \\ & \bar{\mu}_i^j = 0,\ i \in I_2^j, \end{aligned}$$

and our objective is to

$$\text{minimize } c'_1(\bar{x}_1^1 + \ldots + \bar{x}_1^N) + c'_2(\bar{x}_2^1 + \ldots + \bar{x}_2^N)$$

$$\text{subject to: } (\bar{x}_1^j, \bar{x}_2^j, \mu^j) \in (C''_j)$$

$$\lambda_1,\ldots,\lambda_N \ge 0$$

$$\lambda_1 + \ldots + \lambda_N = 1$$

or by doing away with the bars over $\bar{x}_i^j, \bar{\mu}^j$ we reached the equivalent form:

$$\text{minimize } c'_1(x_1^1 + \ldots + x_1^N) + c'_2(x_2^1 + \ldots + x_2^N)$$

$$\text{subject to:}$$

$$\begin{aligned} & \lambda_j d + \mu_1^j a_{21} + \ldots + \mu_m^j a_{2m} = 0 \\ & a'_{1k}x_1^j + a'_{2i}x_2^j = \lambda_j b_i,\ i \in I_1^j \\ & a'_{1i}x_1^j + a'_{2i}x_2^j \le \lambda_j b_i,\ i \in I_2^j \qquad j = 1,2,\ldots,N \\ & \mu_i^j = 0,\ i \in I_1^j \qquad (10) \\ & \mu_i^j \ge 0,\ i \in I_2^j \\ & \lambda_j \ge 0 \\ & \lambda_1 + \ldots + \lambda_N = 1 \end{aligned}$$

which is a Linear Programming problem in the variables $(x_1^1,\ldots, x_1^N, x_2^1,\ldots,x_2^N, \mu^N, \lambda_1,\ldots,\lambda_N)$, $(\mu^j = (\mu_1^j,\ldots,\mu_m^j)')$. It should be noticed that even if some (C'_j) is empty, it will not be so for (C''_j), since (C''_j) always contains the point $\bar{x}_i^j = 0$, $\mu^j = 0$, $\lambda_j = 0$. Also, if $\lambda_j \ne 0$, then (C''_j) and (C'_j) are obviously equivalent. Thus if some (C'_j) is empty the corresponding λ_j will remain 0 for (C''_j) and thus for problem (10).

Although it is clear by now that the Stackelberg problem has been reduced to a classical Linear Programming problem (10), the objection can be raised that the dimensionality of (10) might be quite high (since $N = 2^m$). Nonetheless, while solving (10) it will turn out that most of the x_i^j's, μ_i^j's, λ_j's, involved will be zero during the iterations, as a little reflection will persuade the reader. If this approach of converting the Stackelberg game to an equivalent Linear Programming problem is to be of any value for large m's, we have to find ways to cut down the number of admissible (C'_j)'s. A first step in that direction can be achieved by using Proposition 1 in order to exclude several of the (C'_j)'s. Another thing one could do is to use a combination of the first algorithm presented and of the one presented here as follows: We start by using the first algorithm until a local minimum of (2) is found; (thus Step 3(k) will not be employed). Let us call (x_{1i},x_{2i}), $i = 0,\ldots,\ell$ the points generated. The point $(x_{1\ell},x_{2\ell})$ is a local solution of (2). To check whether it is a global one, we check its stationarity for problem (10). In checking these conditions we do not need to consider those (C'_j) which have as active ones the constraints which are active at any (x_{1i},x_{2i}), $i = 1,\ldots,\ell$. If from $(x_{1,\ell},x_{2,\ell})$ we move to a new — better point — $(x_{1,\ell+1}, x_{2,\ell+1})$ we can emply again the first algorithm until we reach a new local minimum and so on. This combination of the first and the third algorithms amounts to using the form (10) only in order to jump from a possibly local minimum of (2) to a better point.

Having stated the Leader's problem as a Linear Programming problem enables us to solve the Stackelberg problem even if we had employed (2') as the Leader's objective. Obviously this is equivalent to solving

$$\min_{\bar{x}_1^1,\ldots,x_1^N} \left[\max_{x_2^1,\ldots,x_2^N} c'_1(\bar{x}_2^1 + \ldots + x_1^N) + c'_2(x_2^1 + \ldots + x_2^N)\right]$$

subject to the constraints of (10)

This is the problem solved by Falk in [10] which as it was pointed out earlier is equivalent to solving the Stackelberg game with (2) where the Follower's cost is $-c'_2(x_2^1 + \ldots + x_2^N)$ and the Leader's cost is $c'_1(x_1^1 + \ldots x_1^N) + c'_2(x_2^1 + \ldots + x_2^N)$.

DISCUSSION AND CONCLUSIONS

The common goal of the algorithms described here

is to find the global solution of the Stackelberg problem posed in (1)-(2) (or (2')). Although they will achieve that in a finite number of iterations, it is not clear which one is preferable. For example, if one has a good guess about the location of the solution, it seems that algorithm 1 should be preferred. If one knows that the solution is close to the global optimum of the Leader (i.e., the solution of (8), thing which can happen when the direction of d and c_2 are close) then algorithm 2 should be preferred. If one has to terminate the algorithm before the solution is achieved, then algorithm 2 might have not yet achieved to reach a point in $\tilde{\mathcal{R}}$ and thus when we stop we will have not even a suboptimal solution in $\tilde{\mathcal{R}}$; on the contrary, if algorithm 1 is terminated before it finds the global optimum it will still provide us with the last point it had found on $\tilde{\mathcal{R}}$, thing which would guarantee the Leader a certain value for his cost. Thus, if we know a priori that we might have to terminate the algorithm before it finds the global optimum, we might prefer algorithm 1. Of course one can switch back and forth between the first two algorithms. For example, when algorithm 2 provides us with a first point in $\tilde{\mathcal{R}}$ we switch to algorithm 1 starting moving on $\tilde{\mathcal{R}}$ and decreasing the Leader's cost; when we find a local optimum, we find the value of the Leader's cost there and switch back to algorithm 2 where we will use a better u_2^* (see Step k) by considering the value of the Leader's cost at the local optimum as one of the u_1^* over which we minimize to find the new u_2^*.

The algorithms presented can be described in the usual tableau from by setting up the constraints, (3), in standard form. This was not done here, as our aim was to provide the rational and explain the algorithms. Having achieved that, setting up the tableau form is not a difficult task. There are several topics related to the one considered here which merit investigation. For example, that $\tilde{\mathcal{R}}$ is a piecewise linear subset of of boundary of ∂, and the remark iii of Section 4, although are obvious if ∂ is in R^3, require a rigorous proof for more general set-ups. Special techniques are needed as to reduce the computational requirements of the algorithms as well as simulation studies as to gain further insights. Other questions are those which have to do with special schemes for handling degeneracy situations, avoiding the possibility of extensive vertex checking at Step 3 at the first algorithm and simplifying modifications needed if (2') is considered instead of (2) in the third algorithm. Extensions of the algorithms are required in order to handle the case where 3 or more levels of hierarchy exist, or there is one Leader and two Followers which Followers play Nash between themselves.

REFERENCES

[1] H. von Stackelberg, *Marktform und Gleichgewicht*, Springer, Vienna, 1934

[2] C.I. Chen and J.B. Cruz, Jr., "Stackelberg solution for two-person games with biased information patterns," *IEEE Trans. Automat. Contr.*, Vol. AC-17, pp. 791-798, Dec. 1972.

[3] M. Simaan and J.B. Cruz, Jr., "On the Stackelberg strategy in nonzero-sum games," *J. Optimiz. Theory Appl.*, Vol. 11, pp. 533-555, May 1973.

[4] M. Simaan and J.B. Cruz, Jr., "Additional aspects of the Stackelberg strategy in nonzero-sum games," *J. Optimiz. Theory Appl.*, Vol. 11, pp. 613-626, June 1973.

[5] G.P. Papavassilopoulos, and J.B. Cruz, Jr., "Nonclassical control problems and Stackelberg games," *IEEE Trans. Automat. Contr.*, Vol. AC-24, No. 2, pp. 155-166, April 1979.

[6] G.P. Papavassilopoulos and J.B. Cruz, Jr., "Sufficient conditions for Stackelberg and Nash strategies with memory," *J. Optimiz. Theory Appl.*, Vol. 31, No. 2, pp. 233-260, June 1980.

[7] T. Basar and H. Selbuz, "Closed-loop Stackelberg strategies with applications in the optimal control of multilevel systems," *IEEE Trans. Automat. Contr.*, Vol. AC-24, No. 2, pp. 166-179, April 1979

[8] T. Basar, "A general theory for Stackelberg games with partial state information," *Proc. of the Fourth Intern. Conf. on the Analysis and Optimization of Systems*, Versailles, France, Dec. 1980.

[9] Y.-C. Ho, P.B. Luh and R. Muralidharan, "Information structures, Stackelberg games and incentive controllability," *IEEE Trans. Automat. Contr.*, Vol. AC-26, No. 2, pp. 454-460, April 1981.

[10] J.E. Falk, "A linear max-min problem," *Mathematical Programming*, Vol. 5, pp. 169-188, 1973.

[11] G.P. Papavassilopoulos, "Algorithms for Leader-Follower games," *18th Allerton Conf.*, Monticello, Illinois, October 1980.

[12] W.F. Bialas and M.H. Karwan, "Multilevel optimization: a mathematical programming perspective," *19th IEEE-CDC*, Albuquerque, New Mexico, December 1980.

[13] W.F. Bialas and M.H. Karwan, "Two-level linear programming: a primer," Dept. of Industrial Engineering, State Univ. of New York at Buffalo, 1981.

[14] K. Shimizu and E. Aiyoshi, "A new computational method for Stackelberg and min-max problems by use of a penalty method," *IEEE Trans. Automat. Contr.*, Vol. AC-26, No. 2, pp. 460-466, April 1981

[15] J.W. Blankenship and J.E. Falk, "Infinitely constrained optimization problems," *J. Optimiz. Theory Appl.*, Vol. 19, No. 2, pp. 261-281, June 1976.

[16] J. Bracken and J.T. McGill, "Mathematical programs with optimization problems in the constraints," *Operations Research*, Vol. 21, No. 1, pp. 37-44, 1973.