REVUE FRANÇAISE D'AUTOMATIQUE, D'INFORMATIQUE ET DE RECHERCHE OPÉRATIONNELLE. RECHERCHE OPÉRATIONNELLE

R. MATHIEU
L. PITTARD
G. ANANDALINGAM

Genetic algorithm based approach to bi-level linear programming

Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle, tome 28, n° 1 (1994), p. 1-21.

<http://www.numdam.org/item?id=RO_1994__28_1_1_0>

Recherche opérationnelle/Operations Research
(vol. 28, n° 1, 1994, p. 1 à 21)

GENETIC ALGORITHM BASED APPROACH TO BI-LEVEL LINEAR PROGRAMMING (*)

by R. MATHIEU ([1]), L. PITTARD ([2]) and G. ANANDALINGAM ([3])

Communicated by Jacques CARLIER

Abstract. – *This paper reports on the use of a genetic algorithm based technique, GABBA, to solve bi-level linear programming (BLLP) problems. GABBA is used to generate the leader's decision vector, and the follower's reaction is obtained from the solution of a linear program. GABBA is different from the usual genetic algorithms because we only use mutations, alleles of base-10 numbers, and a survival strategy that is suited to BLLP. Results show that, while it takes more cpu time, GABBA gets closer to the global optimum than Bard's [1983] grid search technique for problems of most sizes.*

Keywords: Genetic algorithms, bi-level linear programming, hierarchical optimization.

Résumé. – *Cet article traite de l'utilisation d'une technique à base d'algorithme génétique (GABBA) pour la résolution de problèmes de Programmation Linéaire à Double-Niveau (PLDN). GABBA est utilisé pour générer le vecteur de décision du leader, et la réaction du suiveur est obtenue grâce à la résolution du programme linéaire. GABBA est différent des algorithmes génétiques habituels car il n'utilise que des mutations, des allèles de nombres en base 10, et une stratégie de survie adaptée au PLDN. Les résultats montrent que, bien qu'il demande plus de temps CPU, GABBA s'approche davantage de l'optimum global que la technique d'optimisation par maillage de Bard [1983] pour des problèmes de complexité variable.*

Mots clés : Algorithmes génétiques, programmation linéaire à deux niveaux, optimisation hiérarchique.

1. INTRODUCTION

In this paper, we propose a search technique based on concepts that are non-traditional to the OR community to solve bi-level linear programming

(*) Received April 1992, an earlier version of this paper was presented at the O.R.S.A./T.I.M.S. conference on April 26, 1988 at Washington D. C. We thank, without implicating, the participants of the session on Hierarchical Optimization, and Especially Doug White for comments and criticisms. We also thank an anonymous referee for suggestions for improving the paper.

([1]) Department of Production and Decision Sciences, Cameron School of Business, University of North Carolina at Wilmington, Wilmington, NC 28403 U.S.A.
([2]) Department of Systems Engineering, University of Virginia.
([3]) Department of Systems, University of Pennsylvania.

Recherche opérationnelle/Operations Research, 0399-0559/94/01/$ 4.00

(BLLP) which is a nonconvex problem. Specifically, we present fairly good results of using a modification of the genetic algorithms (Holland, 1975) to solve the BLLP. Bi-level programs (BLP) are a static version of Stackelberg games (von Stackelberg, 1952), that model decision problems involving leader-follower games. Stackelberg games are being extensively analysed in the economics (Laffont and Maskin, 1982), control theory (Ho *et al.*, 1981) and the mathematical programming literature (*see* Anandalingam and Friesz [1992] for a survey). General bi-level programs have been used extensively in the transportation field for network and location planning with integrated supply and user-optimal demand. (*See* for instance, Friesz and Harker [1983], Fisk [1986], LeBlanc and Boyce [1986], and references therein.)

In BLP, the higher level decision maker (the leader) controls decision vector $x \in X \subset R^n$ in order to maximize his objective $F(x, y)$ where $y \in Y \subset R^m$ is the lower level (follower's) decision vector. (Note that for r followers, $y = \{y_1, ..., y_r\}$ and $m = \sum n_i$ where n_i is the dimension of the i-th follower's decision variable). For given $x = x'$, the follower also maximizes his objective function $f(x', y)$ to obtain $y^* = RR(x')$ where $RR(.)$ is the *rational reaction function* of the follower.

Since bi-level mathematical programs are *non-convex*, there is a challenge to find good solution algorithms suited to this particularity. *See* the reviews by Anandalingam and Friesz [1992] and Wen and Hsu [1991]. There are two main approaches to solving bi-level linear programming problems: *Enumeration techniques* including implicit enumeration (Candler and Townsley, 1982), and the "k-th best" algorithm (Bialas and Karwan, 1982) and the *Kuhn-Tucker approaches* which are solved by mixed integer programming (Fortuny-Amat and McCarl, 1982), grid search (Bard, 1983), and parametric complementary pivoting (Bialas and Karwan, 1984). The recent penalty function approach of Anandalingam and White [1988] also belongs to the class of Kuhn-Tucker based algorithms. While these approaches try to handle the nonconvexity, most of them find local rather then global optima, and have no suggestions for dealing with nonunique solutions.

A number of global optimization techniques have been proposed for solving more general class of nonconvex programming problems. They are primarily based on traditional operation research (OR), using techniques such as enumeration of extreme points (Matheiss and Rubin, 1980), cutting-plane and domain partitioning (Glover, 1973), Lagrangian relaxation (Al-Khayyal, 1985), and branch-and-bound (Benson, 1982). The survey by Pardalos and

Rosen [1986] concluded that althrough a variety of methods have been proposed the "few implementable approaches are for functions of special structure, such as quadratic, or separable concave." Also, true to the traditions of operations research, the papers concentrate on analysis of convergence rather than on harnessing the tremendous power of computers to solve these difficult problems. We should note, in passing, that worst case complexity of algorithms has to be analyzed in order to get a feel for computational performance.

The Genetic Algorithm Based Bi-level programming Algorithm (GABBA) we present in this paper also belongs to the class of global optimization techniques. However, it uses a computer-intensive search technique to solve the problem. The *Genetic Algorithm* (GA) bases the search for better solutions on Darwinian principles of looking for fitter genes, where the genes themselves are analogous to admissible solutions to the problem. The population of new solutions are generated from older ones by using genetic operators such as mutation and cross-over, and then tested for fitness. Since the GA follows the generate-and-test paradigm of artificial intelligence (AI), it is clear that GAs would be very appropriate for solving nonconvex programming problems.

GAs and more general adaptive reproductive plans have been applied successfully to many different applications areas. (*See*, for instance, *Proceedings of the Second International Conference on Genetic Algorithms*, 1987.) Genetic algorithms have been used for some optimization problems (DeJong [1980], Goldberg and Richardson [1987] and Abdullah [1991]) including multi-objective learning (Schaffer, 1985). GAs are particularly powerful when applied to problems with complex search spaces (Davis, 1987 *a*). Thus the computational complexity of the BLP suggests that a GA-based approach to solving it would be appropriate.

Since the GA-based heuristic that we propose in this paper falls neither into the vertex enumeration nor Kuhn-Tucker categories, it should be considered a novel approach for BLP problems. In addition, we show that the performance of our GA-based technique compares very favorably to the others suggested in the literature. Our research also contributes to an emerging area of research, that of using artificial intelligence (AI) based techniques to solve operations research problems.

AI based techniques have a great potential in solving difficult operations research problems such as bi-level mathematical programming. In a recent paper, Glover [1987] showed that Tabu Search that he invented provided

much better solution to the *np*-hard Travelling Salesman problem than previous OR based (heuristic) techniques. We should note, however, that the branch-and-cut OR method of Grotschel and Padberg [1979] is the only one that solves the TSP of up to 1,000 cities optimally. Other research in this area includes some initial attempts at linking AI and OR techniques (*see* Glover [1986] and references therein). Useful insights into dynamic programming have also been gained by analyzing it in the context of heuristic search (Kumar and Kanal, 1983). We contribute to this literature as well.

The paper is organized as follows: In the next section, we describe the bi-level linear programming problem, and briefly present the traditional approaches to solve it. Section 3, describes the basic principles behind the genetic algorithm. The genetic algorithm based bi-level programming algorithm (GABBA) is presented in Section 4. We report on computational results in Section 5, and summarize our conclusions in Section 6.

2. THE BI-LEVEL PROGRAMMING PROBLEM

2.1. Overview

Let us consider a two-level hierarchical system where the higher level decision maker (hereafter the "leader") controls decision variables $x \in X$ and the lower level (hereafter the "follower") controls $y \in Y$ respectively. The leader is assumed to select his decision vector first, and the follower select his decision vector after that. In order to formulate the problem, let us define the following:

X=a closed convex set of R^{n1}

Y=a closed convex set of R^{n2}

$f, F : X \times Y \rightarrow R^1$

$g : X \times Y \rightarrow R^m$

where R^j is the real Euclidean space of dimension j. Using this notation, the bi-level programming problem is formulated as:

P1 $$\max_{y \in Y} F(x, y) \quad \text{where } y \text{ solves} \tag{1}$$

$$\max_{x \in X} f(x, y) \tag{2}$$

$$\text{subject to} \quad g(x, y) \leq 0 \tag{3}$$

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

DEFINITION 1: The set $\overline{S}(x) = \{ y : y \in Y, g(x, y) \leq 0 \}$ is called the *Follower's solution set.*

DEFINITION 2: The point-to-set mapping $RR : x \rightarrow R^{n2}$ defined by

$$RR(x) = \{ y^* \in Y : f(x, y^*) \geq f(x, y), \forall y \in \overline{S}(x) \}$$

is called the follower's *rational reaction set.*

Thus the *feasibility set* of problem P1 can be denoted by:

$$S = \{ (x, y) : x \in X, y \in RR(x) \} \tag{4}$$

and the problem can be rewritten as:

P2
$$\underset{x, y}{\text{Max}} \; F(x, y) \tag{5}$$

$$\text{subject to } (x, y) \in S \tag{6}$$

Clearly since, in general, S is a nonconvex set, P2 is a nonconvex programming problem. In the case where all functions are linear, the problem becomes a bi-level linear program which is formulated as follows:

P3
$$\underset{x}{\text{Max}} \; F(x, y) = ax + by \qquad \text{where } y \text{ solves} \tag{7}$$

$$\underset{y}{\text{Max}} \; f(x, y) = cx + dy \tag{8}$$

$$\text{subject to} \quad g(x, y) = Ax + By - p \leq 0 \tag{9}$$

$$x, y \geq 0 \tag{10}$$

Note that once x is given, the follower's objective is simply $\text{Max}_y \; dy$, and cx can be dropped from (8).

2.2. The Bi-Level Programming Literature

The geometric properties of multi-level mathematical programming problems are more complex than familiar mathematical programming problems. Bialas and Karwan [1984] showed that even a simple two-level resource control problem is non-convex. Initial algorithms to solve a special class of *zero-sum* bi-level mathematical programs (BLMP) were based on the branch-and-bound (Falk [1973], Gallo and Ulkucu [1977]) and cutting plane techniques (Konno, 1976).

One of the first solutions for the general problem as formulated in P3 was proposed by Candler and Townsley [1982]. They observed that once an optimal basis to the inside problem was obtained, changing x might affect its feasibility, but not its optimality. Thus they proposed a scheme that involved *implicit enumeration* of adjacent bases to test for feasibility and optimality. Bialas and Karwan [1982] developed a similar *vertex enumeration* procedure

called the "k-th best" algorithm. The leader solves his problem with respect to both leader and follower decision vectors, and would order all basic feasible solutions in such a way that

$$F(x^k, y^k) \geq F(x^{k+1}, y^{k+1}), \quad k = 1, 2, \ldots$$

At any k, given x^k, the follower would solve his problem to obtain $\{y^{*k}\} = RR(x^k)$. If $y^{*k} \neq y^k$, then the algorithm proceeds to the next best solution for the leader, (x^{k+1}, y^{k+1}) and the follower's computation is repeated. Optimality is reached when $y^{*k} = y^k$.

In the *Kuhn-Tucker approach* (Bard and Falk, 1982), the rational reaction set of the follower is replaced by his optimality (Kuhn-Tucker) conditions. The leader takes into account the follower's optimality conditions while solving his own problem; thus the problem can be written equivalently as:

P4

$$\underset{x, y, w}{\text{Max}} \; F(x, y) = ax + by \tag{11}$$

$$\text{subject to} \quad d + w' B = 0 \tag{12}$$

$$w'(Ax + By - p) = 0 \tag{13}$$

$$Ax + By \leq p \tag{14}$$

$$x, y, w \geq 0 \tag{15}$$

where w is the Lagrangian Multiplier associated with equation (3), and the prime stands for transpose.

Attemps at solving problem P4 that results from the Kuhn-Tucker approach includes 0-1 mixed integer programming (Fortuny-Amat and McCarl, 1982), branch-and-bound techniques (Bard and Falk, 1982), grid search (Bard, 1983), and parametric complementary pivoting (Bialas and Karwan, 1984).

2.3. Unresolved Computational Problems

The main computational problem with bi-level mathematical programs is that they are nonconvex (Bialas and Karwan, 1984). Although all approaches up to now have tried to handle the nonconvexity in different ways, with the exception of the k-th best approach, the algorithms find local rather than global optima; the Kuhn-Tucker approach also finds the global optima when Bard's [1983] grid-search method is used. In any case, none of the methods have suggestions for dealing with non unique optima for the follower's problem. Novel approaches and global optimization techniques offer considerable scope in providing more efficient approaches to the problem.

3. THE GENETIC ALGORITHM AND OTHER ADAPTIVE PROCESSES

3.1. Overview

Adaptive processes, of which the genetic algorithm is one, involve the progressive modification of the structure of the process to give better performance for a particular problem environment (Holland, 1975). In genetic algorithms (GAs), the structure is modified according to the general principles of genetics. The GAs are used for global search, but instead of maintaining a single structure (hereafter "solution" to the problem) at any one point in time, it maintains a set of structures (hereafter "population" of solutions). The initial population of solutions is usually generated at random. Each succeeding population of solutions is created from its predecessors using randomized mechanisms which seek to preserve and combine the good characteristics of the better members of the preceeding population of solutions.

Each solution in a Genetic Algorithm is represented as a string of characters from some alphabet, usually ones or zeroes from the binary alphabet. This string is analogous to a chromosome in biological systems and is meant to be sufficient to capture (encode) all the characteristics of the solution it represents. Two genetic operations are typically performed on strings: *Cross-over* replaces part of one string with the corresponding part of another, and *Mutation* arbitrarily changes the value of a character on a string to another member of its alphabet (another of its possible values).

Each candidate solution of each generation is assigned a quantity known as its "fitness" which is the objective function value in optimization problems. Creation of the succeeding generation from its parent generation occurs as follows: To create a member of the succeeding generation, first, a member of the parent generation is selected at random with probability proportional to its fitness divided by the generation average. Then cross-over occurs with probability equal to cross-over probability (a parameter of the GA). If cross-over is not to occur, the single selected member of the parent generation is copied into the succeeding generation intact, at least temporarily. Finally, whether or not cross-over occured, the new member of the succeeding generation is allowed to undergo mutation with (typically low) probability equal to the mutation probalitity (another parameter of the GA). This entire process repeats until the succeeding generation has been "procreated" from the parent generation.

3.2. Optimization Using Genetic Algorithms

From an optimization perspective, the Genetic Algorithm works as follows: The first generation of solutions, generated at random, is uniformly distributed over the search space. Succeeding generations, due to the "survival of the fittest" of their ancestors, tend to be increasingly localized around the best modes in the search space. The final generation usually is concentrated around one or at most few very good modes of the search space. (The number of generations on a GA, as well as the generation size, is an exogenous parameter, usually sufficiently large for the preceding to occur.)

The general genetic algorithm can be described as follows (Grefenstette, 1986):

```
k = 0
Initialize P(k)–the population at generation k
Evaluate P(k)
While termination condition not satisfied, Do
Begin
    k = k + 1
    Select P(k)
    Recombine P(k)
End
```

It should be noted here that the genetic algorithm, and GABBA, our GA-based optimization algorithm, may be better described as a "heuristic" rather than an "algorithm", especially in the vernacular of operations research analysts. We will continue to use "algorithm" since it is standard practice in the GA field, and also fits well within the use of the word by computer scientists.

4. THE GABBA ALGORITHM

4.1. Basic Parameters

The Genetic Algorithm Based Bi-level programming Algorithm (GABBA) is best described as an adaptive reproductive plan based on some principles of the genetic algorithm. In order to solve the bi-level linear program, the leader's decision vector is reproduced according to a modification of the GA, and the follower's decision vector is obtained by solving the second level linear programming problem. The fitness test also involves a modification of traditional GA tests. GABBA is described in detail in the next subsection. In this subsection, we will highlight the basic parameters of GABBA.

GABBA's reproductive plan is controlled by the operators, population size, number of structures in the previous generation's population to reproduce,

precentage of alleals in each structure to reproduce, number of new structures to produce randomly every generation, and selection strategy. It does not exploit cross-over. It could be argued that cross-over is a special case of mutation, since it could be reduced to multiple mutations. Many argue that crossover is essential in GA, as it is responsible for propagating the characteristics of best structures, like real-life genetic processes (this was argued by a referee; *see* also Holland [1975]). It can be argued that, in general, using mutations primarily does not allow each successive generation to be strongly connected to the parent generation. This is not true in our case, because we use mutations to change only part of the parent characteristics. Inspite of the fact that we do not use cross-over, all our attempts at producing populations of solutions, reducing population size etc. places us closer to a GA than a pseudo random search algorithm.

We should note here that, unlike in most pure genetic algorithms, GABBA does not encode its structures as 0-1 bit strings, but rather as a string of base-10 digits. The new code is more intuitive for adapting reproductive plans to mathematical programming.

In GABBA, the *population size N* is the number of vectors z, where $z=(x, y)$ is the overall solution; *i. e.* both leader and follower decision vectors to the problem. In relation to the definition of the bi-level programming problem given in section 2, $z \in Z \subset R^{n1+n2}$. Individual members in the population (which we will call *structures*), are represented as points (*i. e.* vectors) in $(n1+n2)$-dimensional space. We represent the population at generation (iteration) k, by $P(k)$. As in traditional genetic algorithms, we will call each component of the structure an *alleal.* We provide an example to clarify the notation:

Example 1: Suppose we have the following population.

Population #	x_1	x_2
1	2.2	3.1
2	5.6	3.7
3	7.8	10.9

There are 3 structures, and for structure #2, the alleals are 5.6 and 3.7. Structures #2 itself will be given by the 2-dimensional vector (5.6, 3.7). ■

In GABBA, all structures are defined by alleals of base-10 numbers. the base-10 adaptive process is fundamental to GABBA, and is presented in

detail here. Each alleal is defined to have a head and a tail. The point that separates the head from the tail is determined by a parameter, SCALE, where

$$\text{SCALE} = 10^n, \quad n \in I$$

where I is the set of positive real integers. For example, if our alleal is 12345.67, and SCALE=10.0, the head of the alleal is 1234*.**, and the tail of the alleal is ****5.67.

When optimizing a function using this base-10 scheme, the information to be passed on to the next generation is contained in the head. When creating a new species from an old species, the head of the previous generation is kept, and a random number, generated using a uniform distribution on the interval (0, SCALE) becomes the new tail. For example, if 1234*.** is passed to the new generation, and a random tail of 9.81 is generated, the new species becomes 12349.81. Thus, we keep generating tails at a given SCALE level until "good" solutions are obtained. After we determine what are "good" solutions based on the problem objective, the SCALE value is modified so that new solutions are more precise than the old ones. One heuristic to revise the SCALE parameters, after determination of "good" solutions, could be:

$$\text{(new) SCALE} = \text{(old) SCALE}/10. \tag{16}$$

The population is generated within feasible lower and upper bounds. Thus the *alphabet* for each alleal is all base-10 numbers between the lower and upper bounds; this is different from traditional uses of GAs, where the alphabet is made up of the binary code (0, 1). If the SCALE is changed according to equation (16), we will have as many macro-iterations in GABBA as the order of magnitude that the alphabet spans.

The *selection strategy* is based on obtaining, at generation, the N most fit structures (*i.e.* solutions with the highest value of $F(x, y)$, the leader's objective). Candidates for selection include the N structures from the previous generation, NP offspring structures newly created by mutation of structures in the previous generation, and NR new structures produced randomly. Note that not all NR candidates for mutation will produce feasible solutions. Thus overall there will be slightly *less* than $(N+NP+NR)$ candidates from which to select the N best for the next generation.

4.2. The Algorithm

We will now present GABBA in detail. In order to keep the description concise, we will adopt the following notation:

I, set of positive real numbers;

int(y), integer value of positive real number y;

$U[a, b]$, uniform distribution between a and b;

$x(k)$, $y(k)$, leader and follower decision vectors at generation k;

x_i, i-th component of the leader's decision vector x;

$$x = \{x_1, \ldots, x_{n1}\};$$

y_i, i-th component of the follower's decision vector y;

$$y = \{y_1, \ldots, y_{n1}\};$$

x^j, y^j, j-th structure of leader and follower decision vector;

x_i^j, y_i^j, i-th component of j-th structure of vectors x and y;

$F^j(k)$, the value of the leader's objective given by equation (1) for $x=x^j$, $y=y^j$ at the k-th generation;

$P(k)$, the population at generation k: $P(k)=\{x(k), y(k)\}$.

Note that in the case of the decision vectors and their components, although we suppress k, the generation index, it should be understood that we are considering these vectors at the appropiate generation.

The reader should note that, as given by Definition 2 in section 2, whenever we refer to the follower's rational reaction set $RR(x)$, we mean that the following linear program is solved, *for given* x, (say $\hat{x}$), using a standard algorithm based on the simplex method:

P5
$$\text{Max}_y \; dy$$
$$\text{s. t.} \quad By \leq p - A\hat{x}$$
$$y \geq 0$$

The GABBA algorithm proceeds as follows:

Step 0 (Initialization):

Let $k=0$, $F^*(-1)=-\infty$.

Set parameters:

(*a*) *N*-population size;

(*b*) *NP*-number of current solutions (*i.e.* structures) in population $P(k)$ to undergo mutation;

(*c*) *NX*-number of decision variables (*i.e.* alleals) x to undergo mutation;

(*d*) *NR*-number of new random solutions created during each iteration;

(*e*) ϵ-degree of accuracy required.

Step 1 (Set Bounds):

Generate an upper bound (and lower bound) for each $x_i \in x$ by solving the following problem for $i=1, ..., n_1$:

P6
$$\underset{x,y}{\text{Max (Min)}}\, x_i$$
$$\text{s.t.} \quad Ax + By \leq p$$
$$x = (x_1, ..., x_{n1})$$
$$x, y \geq 0$$

Set SCALE$=10^n$, $n \in I$, such that

$$\text{SCALE} \geq \text{Max}\,(x_1, x_2, ..., x_{n1})$$

Step 2 (Generate Initial Population):

$$j = 1$$

Generate population $P(k)$, which contains N vectors

$$z^j = \{x^j, y(x^j)\}, \qquad j = i, ..., N,$$

as follows:

(2.1) $x_i^j \simeq U[\text{Min } x_i^j, \text{Max } x_i^j]$, $i = 1, ..., n_1$,

(2.2) Solve P5:

(*a*) If feasible, obtain $y(x^j) \in RR(x^j)$

If $j=N$, go to Step 3.

Else, set $j=j+1$.

Go To Step 2.1.

(*b*) If infeasible, discard x^j, go to Step 2.1.

Step 3 (SCALE Modification):

Sort array $P(k)$ according to Level 1 objective:

$$F^j(k) = ax^j + by(x^j), \qquad j = 1, ..., N.$$

[*i.e.* $F^j(k) \geq F^{j-1}(k)$, $\forall j$].

Let $z^j = (x^j, y^j) = \arg F^j(k)$, $j = 1, ..., N$.

Store $F^*(k) = \text{Max } F^j(k)$, and $\{x^*(k), y^*(k)\} = \arg F^*(k)$.

Let $F_m(k) = \sum_{j=1}^{mN} F^j(k)/m.N.$

If $F^*(k) = F^*(k-1)$ *and*

If $F_{0.6}(k) \geqslant 0.85.F^*(k)$.

Then SCALE = SCALE/10.

Step 4 (Stopping Criterion):

If SCALE $< \epsilon$, Then STOP.

Satisfactory solution is $\{x^*(k), y^*(k)\}$.

Otherwise, Go to Step 5.

Step 5 (Mutated Structures):

$$j = 0$$

Let $z_1^j = \{x^j, y(x^j)\}$, $j = 1, ..., NP$.

(Note $NP < N$, and $\{z_1^j\} \subset \{z^j\}$)

Mutate the NP vectors z_1^j as follows:

(5.1) $j = j + 1$.

Obtain $x^{j'}$ by mutating NX randomly selected alleals of x such that

$$x_i^{j'} = \text{int}\{x_i^j/\text{SCALE}\} * \text{SCALE} + w,$$

with

$$w \simeq U[0, \text{SCALE}]$$

(5.2) Solve P5

(i) if solution feasible, $y(x^{j'}) \in RR(x^{j'})$.

If $j = NP$, go to Step 6.

Else, go to Step 5.1.

(ii) if solution infeasible, discard x^j,

Go to Step 5.1.

Step 6 (New Random Structures):

$$j = 0$$

Let $z_2^j = \{x^j, y(x^j)\}$, $j = 1, ..., NR < N$.

Generate NR vectors z_2^j as folows:

(6.1) $j = j + 1$

$$x_i^j \simeq U[\text{Min } x_i^j, \text{Max } x_i^j], \qquad i = 1, ..., n_1,$$

(6.2) Solve P5:

(i) If feasible, obtain $y(x_i) \in RR\ (x^j)$.

If $j=RN$, go to Step 7.

Else go to Step 6.1

(ii) If infeasible, discard x^j, go to Step 6.1.

Step 7 (Selection):

From N, $\{x, y(x)\}$ structures from $P(k)$, and NP or less, $\{x', y(x')\}$ structures from mutation, and NR, $\{x'', y(x'')\}$ population from random structures.

Select

$P(k+1) = \{N$ structures that have highest value of $F(.)\}$

$$k = k + 1$$

Go to Step 3.

4.3. Convergence and Stopping Rules

Many probabilistic search techniques have been proposed in the operations research literature (*see* Rinooy Kan and Timmer [1987]). All these methods consider the problem to be solved if, for some $\epsilon > 0$, an element of the following set has been identified:

$$A_z(\varepsilon) = \{z \in Z : \|z - z^*\| \leq \varepsilon\} \tag{20}$$

$$A_F(\varepsilon) = \{z \in Z : |F(z) - F(z^*)| \leq \varepsilon\} \tag{21}$$

In step 5 of GABBA, we use the set $A_z(\epsilon)$ for each SCALE level of the algorithm. Clearly at very large values of SCALE, it dominates ϵ and we move to the next level only when z exactly equals z^*. At lower values of SCALE, the stopping value ϵ would tend to dominate.

Generally one cannot guarantee *absolutely* that the probabilistic methods would provide a solution $z \in A$. Under conditions on the sampling distributions and the curvature of $F(.)$, it can be proved that an element of A is sampled *almost surely* as the sample size increases. One option to absolutely guarantee success would be to use the probabilistic technique in the first stage for locating regions where good local optima exist, and to use traditional OR techniques at the second stage to find the actual optima. We do not do this in this paper.

5. COMPUTATIONAL RESULTS

5.1. Preliminaries

A random bi-level linear programming problem generator was constructed in order to test GABBA against other algorithms. The problems generated were of size (3, 7, 4), (5, 10, 6), (6, 14, 8), and (8, 17, 10), where (n_1, n_2, m) represents the number of the level 1 decision variables, the level 2 decision variables, and the constraints respectively. The problems contained both positive and negative coefficients in the objectives and constraints. The coefficients matrix was kept full by allowing only a small percentage of 0's. In all 15 random problems were generated for each problem size. The problems were run on an AT&T PC6300 Plus microcomputer with Intel 80286 microprocessor, and 80287 math coprocessor.

5.2. Bard's Grid Search Technique

The performance of GABBA was tested against Bard's [1983] grid search technique. The choice of the grid search technique was based on the fact that neither it nor GABBA guarantee global optimality, and the former technique has had much publicity in the literature (*see* LeBlanc and Boyce, 1985), and is generally acknowledged as an efficient algorithm.

Bard [1983] showed that bi-level linear programs can be solved by obtaining a maximal λ (say $\lambda_{\max}$) that solves the following straightforward linear program:

$$\left.\begin{aligned} P(x, y, \lambda) = \underset{x, y, \lambda \geq 0}{\text{Max}} \; & \lambda (ax + by) + (1 - \lambda)(dy) \\ \text{s.t.} \quad & Ax + By \leq p \end{aligned}\right\} \tag{22}$$

and provides a $y \geq 0$ that is feasible for the problem:

$$Q(y) = \text{Max } dy \quad \text{s.t. } By \leq p - Ax \tag{23}$$

Bard's procedure is to iteratively solve the problem $P(x, y, \lambda)$ for varying values of λ, where new values of λ are found from sensitivity analysis concepts, and arranged on a grid. (*See* Bard [1983] for details.) The modified grid search (MGS) algorithm used in this paper follows the general principle in Bard's [1983] paper, but uses upper and lower bounds on λ at each iteration so as to converge to $\lambda_{\max}$ quickly. The algorithm is stopped when there is no change in the λ's.

5.3. Computational Results

Performance was measured on the basis of computational effectiveness, measured by CPU time, and quality of solution, measured by two means:

(i) the percent of time that one of the algorithms succeeded in producing a better solution (*i. e.* a higher level 1 objective at the end of it's execution), and (ii) the sum of absolute deviation of the best solutions of each defined by

$$\Delta = \sum_{m=1}^{M} \{F_m^*(\text{MGS}) - F_m^*(\text{GABBA})\} \tag{24}$$

where F_m^* is the best value of F obtained from the n-th problem, and M is the total number of problems. Note that if the absolute deviation Δ is positive, it means that the solutions produced by the Grid Search technique were better than those produced by GABBA in an overall sense. Conversely, if Δ was negative, then GABBA was better overall than the Grid Search technique.

For problems of two sizes, GABBA was run for varying setting of *NR* (number of decision vectors to obtain from new random generation) and *NX* (number of alleals to undergo mutation). The results are shown in tables I and II and the parameter settings are shown in table III. The settings of *NP* (number of current structures to undergo mutation), *N* (population size), and the SCALE modification heuristic (*see* Step 1 above) were held constant. Specifically, we set $N=100$ (also $n_1=50$), and $NP=40$. Also, the stopping criterion ϵ was set at 10^{-4}. From the results in tables I and II, it is not clear what the best GABBA algorithm is. When measured in terms of average cpu time (in seconds), GABBA1 seems to be best in both problem instances. However, compared to the grid search technique, GABBA1 only yields a better near optimal solution in 33.3 per cent of the time for problem size (3, 7, 4), and 50.0 per cent of the time for problem size (5, 10, 6). GABBA5 yields the near optimal solution 67 per cent of the time for problem size (5, 10, 6) but has almost a 70 per cent higher cpu time compared to GABBA1. We decided that the probability of reaching a near optimal solution was more important than cpu time. Hence, GABBA5, with $NR=0.5N$, $NX=0.5n$, $NP=0.4N$, and $N=2n$, was chosen for all subsequent analyses.

It should be noted however, that although we have experimented with some of the parameters of GABBA, we have not optimized them in the sense of Grefenstette's [1986] paper. The vast number of computer runs required for optimizing parameters was beyond our research budget. The more important reason for not performing the numerous computations was that experimentation seemed to show that the optimal parameter setting were very sensitive to problem size, and other problem characteristics. Thus, a strict optimization of parameters may not have been possible.

The fairly simple bi-level linear programming technique based on Genetic Algorithms provided reasonably good results. Although the grid search

TABLE I

GABBA results for different parameter settings. Problem size (3, 7, 4)

Algorithm	E [cpu]	Var [cpu]	% Better	Δ
Grid Search . . .	4.24	11.34	n. a.	n. a.
GABBA1	17.13	56.31	33.3%	0.556
GABBA2	26.61	135.30	38.0%	0.435
GABBA3	26.56	134.63	38.0%	0.435
GABBA4	20.13	105.40	44.0%	0.160
GABBA5	24.81	176.70	40.0%	0.181
GABBA6	24.77	176.79	40.0%	0.181
GABBA7	20.88	105.41	44.4%	0.160
GABBA8	24.81	177.58	40.0%	0.181
GABBA9	24.76	176.95	40.0%	0.181

TABLE II

GABBA results for different parameter settings. Problem size (5, 10, 6)

Algorithm	*E* [cpu]	Var [cpu]	% Better	Δ
Grid Search . . .	13.68	72.89	n. a.	n. a.
GABBA1	47.76	388.17	50.0%	-0.029
GABBA2	67.19	1531.58	38.0%	0.144
GABBA3	82.68	1562.97	38.0%	0.295
GABBA4	52.87	764.01	44.0%	0.153
GABBA5	80.48	2,092.73	67.0%	-0.057
GABBA6	82.55	1,880.62	57.0%	0.075
GABBA7	70.02	1,071.84	40.0%	0.247
GABBA8	99.83	2,832.03	57.0%	-0.026
GABBA9	110.77	3,163.03	57.0%	0.063

algorithm took less time to solve the problems of different size, GABBA yielded a better solution most of the time (table IV). In problems of size (6, 14, 8), GABBA was 73 out of 100 times better than the grid search technique. This means that GABBA would be more likely to reach a global optimum than would grid search, or, by the same token, any other technique proposed to solve bi-level linear programs (*see* Bialas and Karwan, 1984). Of course, as in most pseudo-random search techniques, the computation time was very long. This is a major disadvantage of these methods.

One reason for the slowness could be the fact that we did not exploit cross-over. In addition, the stopping rule for GABBA was more stringent than for the grid search method. These modifications are left for future research. However, we can conclude that genetic algorithms have a great potential for solving bi-level programming and other nonconvex programming problems.

TABLE III
Parameter settings for the different algorithms

Algorithm	*NR*	*NX*
GABBA1	0.2 N	$0.2\,n_1$
GABBA2	0.2 N	$0.5\,n_1$
GABBA3	0.2 N	$0.8\,n_1$
GABBA4	0.5 N	$0.2\,n_1$
GABBA5	0.5 N	$0.5\,n_1$
GABBA6	0.5 N	$0.8\,n_1$
GABBA7	0.8 N	$0.2\,n_1$
GABBA8	0.8 N	$0.5\,n_1$
GABBA9	0.8 N	$0.8\,n_1$

Note: All results were rounded to the nearest integer.

TABLE IV
GABBA versus grid search

Problem size	Grid search *E* [cpu]	GABBA *E* [cpu]	GABBA dominates grid search (%)	Δ
(3, 7, 4)	4.23	24.81	40.0%	0.181
(5, 10, 6)	13.69	80.48	66.6%	–0.057
(6, 14, 8)	27.27	699.45	73.0%	–0.135
(8, 17, 10)	63.99	3,281.59	60.0%	–0.593

6. CONCLUDING REMARKS

In this paper, we reported on the use of GABBA, a technique based on the genetic algorithm to solve the bi-level linear programming problem. We found that the technique was easy to use, and was not affected by the nonconvexity of the problem, of the nonuniqueness of the solution. The latter properties are very attractive because most algorithms that rely on smoothness properties of the objective functions and feasibility regions (*i. e.* variations of the steepest descent or ascent) perform quite badly on domains that are nonconvex. We showed in this paper that by using an intelligent way, based on the genetic algorithm, to generate new solutions to test for suitability (or near-optimality), we could overcome much of the barriers to solving nonconvex programming problems. While, as in most probabilistic search techniques, we could not guarantee convergence, GABBA found solutions

that were closer to the global optimum than the well known Kuhn-Tucker method solved by Bard's [1983] grid search technique.

The main problem with GABBA was that the computational time was long, especially when we used a microcomputer. Given the recent revolution in computational power, both memory and speed, and the fact that microcomputers are readily available to individual researchers, we would argue that using computer-intensive search techniques would not be an inefficient use of resources. The time spent on obtaining mathematically elegant proofs of convergence for algorithms that eventually do not yield the global optimum may be better spent by microcomputers searching for the solution. This is the basic premise of much of the recent work on the use of artificial intelligence techniques to solve difficult operations research problems such as the travelling salesman problem.

However we should not minimize the importance of establishing theorems to bound the worst-case performance of an OR heuristic. Most difficult optimization or decision problems are *NP*-complete, and the use of *AI* based techniques does not suppress this unpleasant property. It is important to derive algorithms with polynomial time complexity to solve these problems. It is vital to show what the worst case behavior of any proposed algorithm is. We have not done so in this paper, and leave it as work for future research.

REFERENCES

A. R. Abdullah, A Robust Method for Linear and Nonlinear Optimization Based on Genetic Algorithm *Cybernetica*, 1991, *34*, No. 4, pp. 279–287.

F. A. Al Khayyal, Minimizing A Quasiconcave Function Over a Convex Set: A Case Solvable by Lagrangian Duality, *Proceedings*, I.E.E.E. International Conference on Systems, Man, and Cybernetics, Tucson, 1985, AZ, pp. 661-663.

G. Anandalingam, A Mathematical Programming Model of Decentralized Multi-Level Systems, *J. of the Operational Research Society*, 1988, *39*, No. 11.

G. Anandalingam and D. J. White, A Penalty Function Approach to Bi-Level Linear Programming, working paper, Department of Systems, University of Pennsylvania, August, 1988.

G. Anandalingam and T. L. Friesz, Hierarchical Optimization: An Introduction, *Annals of Operations Research*, 1992, *34*, pp. 1-11.

J. F. Bard, An Efficient Point Algorithm for a Linear Two-Stage Optimization Problem, *Operations Research*, 1983, July-August, pp. 670-684.

J. F. Bard and J. E. Falk, An Explicit Solution to the Multi-Level Programming Problem, *Computers and Operations Research*, 1982, *9*, No. 1, pp. 77-100.

H. P. Benson, On the Convergence of Two Branch and Bound Algorithms for Nonconvex Programming, *J. of Optimization Theory and Application*, 1982, *36*, pp. 129-134.

A. D. Bethke, Genetic Algorithms as Function Optimizers, *Ph. D. dissertation*, unpublished, University of Michigan, Ann Arbor, 1981.

W. F. Bialas and M. H. Harwan, On Two-Level Optimization, *I.E.E.E. Transactions on Automatic Control*, 1982, *AC-27*, pp. 211-214.

W. F. Bialas and M. H. Karwan, Two-Level Linear Programming, *Management Science*, 1984, *30*, No. 8, August, pp. 1004-1020.

W. Candler and R. Townsley, A Linear Two-Level Programming Problem, *Computers and Operations Research*, 1982, *9*, No. 1, pp. 59-76.

L. Davis (Ed.), *Genetic Algorithms and Simulated Annealing*, Morgan Kaufman Publishers, Los Altos, CA, 1987 *a*.

L. Davis, *Performance of a Genetic Algorithm on the Network Link Size Problem*, paper presented at the O.R.S.A./T.I.M.S. meeting, St. Louis, October, 1987 *b*.

K. De Jong, Adaptive System Design: A Genetic Approach, *I.E.E.E., Transactions on Systems, Man, and Cybernetics*, 1986, *16*, Jan.-Feb.

J. E. Falk, A Linear Mini-Max Problem, *Mathematical Programming*, 1973, pp. 169-188.

C. S. Fisk, A Conceptual Framework for Optimal Transportation Systems Planning with Integrated Supply and Demand Models, *Transportation Science*, 1986, *20*, No. 1, pp. 37-47.

J. Fortuny-Amat and B. McCarl, A Representative and Economic Interpretation of a Two Level Programming Problem, *J. of the Operational Research Society*, 1981, *32*, pp. 783-792.

T. L. Friesz and P. T. Harker, Multicriteria Spatial Price Equilibrium Network Design: Theory and Computational Results, *Transportation Research*, 1983, *17b*, pp. 203-217.

G. Gallo and A. Ulkucu, Bi-Linear Programming: An Exact Algorithm, *Mathematical Programming*, 1977, *12*, pp. 173-194.

F. Glover, Convexity Cuts and Cut Search, *Operations Research*, 1973, *21*, pp. 123-134.

F. Glover, Future Paths for Integer Programming and Links to Artificial Intelligence, *Computers & Operations Research*, 1986, *13*, (5). pp. 533-549.

F. Glover, *Tabu Search*, mimeo, Center for Applied Artificial Intelligence, Graduate School of Business, University of Colarado, October, 1987.

D. E. Goldberg and J. Richarsdon, Genetic Algorithms with Sharing Multi-Modal Function Optimization, in *Genetic Algorithms and Their Application: Proceedings of the Second International Conference on Genetic Algorithms*, M.I.T., Cambridge, MA, 1987.

J. Grefenstette, Optimization of Control Parameters for Genetic Algorithms, *I.E.E.E. Transactions on Systems, Man, and Cybernetics*, 1986, *16*, (1), January-February, pp. 122-128.

M. Grotschel and M. W. Padberg, On the Symmetric Travelling Salesman Problem I: Inequalities, II: Lifting Theorems and Facets, *Mathematical Programming*, 1979, *16*, pp. 265-302.

Y. C. Ho, P. B. Luth and R. Muralidharan, Information Structures, Stackelberg Games, and Incentive Controllability, *I.E.E.E. Transactions on Automatic Control*, 1981, *AC-31*, No. 4, pp. 670-684.

J. H. Holland, *Adaption in Natural and Artificial Systems*, The University of Michigan, 1975, Ann Arbor, MI, 1975.

S. Kirkpatrick, *Combinatorial Search Using Simulated Annealing*, paper presented at the O.R.S.A./T.I.M.S. meeting, St. Louis, October, 1987.

H. Konno, A Cutting Plane Algorithm for Solving Bilinear Programs, *Mathematical Programming*, 1976, *11*, pp. 14-27.

V. Kumar and L. N. Kanal, *Some New Insights into the Relationships Among Dynamic Programming, Branch and Bound, and Heuristic Search Procedures*, Proceedings, I.E.E.E. International Conference on Systems, Man, and Cybernetics, 1983, pp. 19-23.

J.-J. LAFFONT and E. MASKIN, The Theory of Incentives: An Overview, in W. HILDENBRAND Ed., *Advances in Economic Theory*, Cambridge University Press, Cambride, U.K., 1982, pp. 31-94.

L. J. LEBLANC and D. E. BOYCE, A Bi-Level Programming Algorithm for Exact Solution of the Network Design Problem with User-Optimal Flows, *Transportation Research B*, 1986, *28*, pp. 259-265.

T. H. MATHEISS and D. S. RUBIN, A Survey and Comparison of Methods for Finding All Vertices of Convex Polyhedral Sets, *Mathematics of Operations Research*, 1980, *5*, pp. 167-185.

P. M. PARDALOS and J. B. ROSEN, Methods for Global Concave Minimization: A Bibliographic Survey, *S.I.A.M. Review*, 1986, *28, (3)*, pp. 367-379.

A. H. G. RINNOOY and G. T. TIMMER, Stochastic Global Optimization Methods, part I: Clustering Methods and part II: Multi-level, Methods *Mathematical Programming*, 1987, *39*, No. 1, pp. 27-78.

J. D. SCHAFFER, Learning Multiclass Pattern Determination, in *Proceedings of the International Conference on Genetic Algorithms and Their Applications*, Robotics Institute of Carnegie-Mellon University, Pittsburg, PA, 1985.

H. von STACKELBERG, *The Theory of the Market Economy*, Oxford Univesity Press, Oxford, 1952.

U. P. WEN and S. T. HSU, Linear Bi-level Programming: A Review, *J. of the Operational Research Society*, 1991, *42*, No. 2, pp. 125-133.