Available online at www.sciencedirect.com

ScienceDirect

European Journal of Operational Research 188 (2008) 14–28

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/ejor

Continuous Optimization

A new approach for solving linear bilevel problems using genetic algorithms ☆

Herminia I. Calvete [a,*], Carmen Galé [b], Pedro M. Mateo [a]

[a] *Dpto. de Métodos Estadísticos, Universidad de Zaragoza, Pedro Cerbuna 12, 50009 Zaragoza, Spain*
[b] *Dpto. de Métodos Estadísticos, Universidad de Zaragoza, María de Luna 3, 50018 Zaragoza, Spain*

Received 20 January 2006; accepted 27 March 2007
Available online 21 April 2007

Abstract

Bilevel programming involves two optimization problems where the constraint region of the first level problem is implicitly determined by another optimization problem. This paper develops a genetic algorithm for the linear bilevel problem in which both objective functions are linear and the common constraint region is a polyhedron. Taking into account the existence of an extreme point of the polyhedron which solves the problem, the algorithm aims to combine classical extreme point enumeration techniques with genetic search methods by associating chromosomes with extreme points of the polyhedron. The numerical results show the efficiency of the proposed algorithm. In addition, this genetic algorithm can also be used for solving quasiconcave bilevel problems provided that the second level objective function is linear.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Bilevel programming; Genetic algorithm; Extreme point

1. Introduction to bilevel programming

Bilevel programming has been proposed for dealing with decision processes involving two decision-makers with a hierarchical structure. The second level decision-maker optimizes his/her objective function under the given parameters from the first level decision-maker. This one, in return, with complete information on the possible reactions of the second level decision-maker, selects the parameters so as to optimize his/her own objective function. Hence, bilevel problems are characterized by the existence of two optimization problems in which the constraint region of the first level problem is implicitly determined by another optimization problem. In other words, bilevel problems have a subset of variables constrained to be an optimal solution of another problem parameterized by the remaining variables. General bilevel problems can be formulated as

$$\begin{aligned}&\max_{(x_1,x_2)\in S} f_1(x_1,x_2),\\ &\text{where } x_2 \in \arg\max_{v\in S(x_1)} f_2(x_1,v),\end{aligned} \tag{1}$$

☆ This research work has been supported by the Spanish Ministry of Education and Science under grant MTM2004-00177.
* Corresponding author. Tel.: +34 976 762210.
E-mail addresses: herminia@unizar.es (H.I. Calvete), cgale@unizar.es (C. Galé), mateo@unizar.es (P.M. Mateo).

where $x_1 \in \mathbb{R}^{n_1}$ and $x_2 \in \mathbb{R}^{n_2}$ are the variables controlled by the first level and the second level decision-maker, respectively; $f_1, f_2 : \mathbb{R}^n \to \mathbb{R}, n = n_1 + n_2$; $S \subset \mathbb{R}^n$ defines the common constraint region and $S(x_1) = \{x_2 \in \mathbb{R}^{n_2} : (x_1, x_2) \in S\}$. Due to their structure, bilevel programs are nonconvex and quite difficult to deal with. Bard [2] and Dempe [6] are good general references on this topic. Besides them, Dempe [7] provides a survey which covers applications as well as major theoretical developments.

Despite the wide range of real systems in which a bilevel programming problem would be appropriate to model the decision process, the truth of the matter is that applications of bilevel programming are scarce. One of the main factors that account for this fact is the lack of efficient algorithms for solving bilevel problems. The characteristics of bilevel problems, mainly the nonconvexity, cause difficulties, even when all involved functions are linear. In fact, linear bilevel problems are NP-hard. To date, a variety of exact algorithmic approaches have been developed. Some involve the use of enumerative schemes, whereas others replace the second level problem with its Karush–Kuhn–Tucker (KKT) conditions or apply penalty function or gradient methods, etc. But, all of them are very time consuming, especially when the problem is large. For these kind of problems, metaheuristic algorithms seem to be necessary. These have proved to be robust in finding good solutions to complex optimization problems and are able to solve large problems in reasonable computational time.

In this paper we focus on the linear bilevel programming (LBP) problem, a special case of (1), in which functions f_1 and f_2 are linear and S is a polyhedron. Taking into account the existence of an extreme point of the polyhedron S which solves the LBP problem [3], the aim of this paper is to combine classical enumeration techniques which search for extreme points of S with metaheuristic methods. Particularly, in this paper we combine the underlying framework of genetic algorithms with a sort of extreme point enumeration to develop an algorithm providing near-optimal solutions in acceptable computational times. For this purpose, the proposed chromosomes represent feasible basic solutions of the polyhedron S and crossover, mutation, evaluation and selection operators are properly defined. The paper is organized as follows. Section 2 sets the problem and outlines major ideas which have been used to develop algorithms for solving LBP problems. Section 3 proceeds by developing the algorithm GABB (Genetic Algorithm Based on Bases) for solving the LBP problem. Also, some theorems are proved which allow us to improve the process of chromosomes fitness evaluation. In Section 4 the computational performance of the procedure is evaluated. Finally, Section 5 concludes the paper with final remarks on more general bilevel problems for which the algorithm is still valid and some ideas on future work.

2. Background

Using the common notation in bilevel programming, the LBP problem can be written as follows:

$$\begin{array}{lll} \max\limits_{x_1,x_2} & f_1(x_1,x_2) = c_1x_1 + c_2x_2, & \text{where } x_2 \text{ solves} \\ \max\limits_{x_2} & f_2(x_1,x_2) = d_2x_2 & \\ \text{s.t.} & A_1x_1 + A_2x_2 \leqslant b, & \\ & x_1, x_2 \geqslant 0, & \end{array} \tag{2}$$

where c_1 is an n_1-dimensional row vector, c_2 and d_2 are n_2-dimensional row vectors, A_1 is an $m \times n_1$-matrix, A_2 is an $m \times n_2$-matrix and b is an m-dimensional column vector. We assume that the polyhedron S defined by the common constraints is nonempty and bounded.

Let S_1 be the projection of S on $\mathbb{R}^{n_1}$. For each $x_1 \in S_1$, a feasible solution to the LBP problem (2) is obtained by solving the following linear programming problem:

$$\begin{array}{ll} \max\limits_{x_2} & d_2x_2 \\ \text{s.t.} & A_2x_2 \leqslant b - A_1x_1, \\ & x_2 \geqslant 0. \end{array} \tag{3}$$

Let $M(x_1)$ denote the set of optimal solutions to (3). We assume that for all decisions taken by the first level decision-maker, the second level decision-maker has some room to respond, i.e. $M(x_1) \neq \emptyset$. Hence, the feasible region of the first level decision-maker, called inducible region IR, is

$$\text{IR} = \{(x_1, x_2) : x_1 \in S_1, x_2 \in M(x_1)\}.$$

Any point of IR is a bilevel feasible solution. Moreover, to ensure that the LBP problem is well posed we assume that $M(x_1)$ is a point-to-point map [2,6]. Amongst the approaches which have been proposed for solving LBP problems, probably the two most popular techniques are enumeration and the use of

KKT conditions. On the one hand, enumeration methods make use of the fact that an extreme point of S exists which is an optimal solution of the LBP problem. On the other hand, if the second level problem is replaced by its KKT conditions, the LBP problem is transformed into a nonlinear one-level problem. So, special techniques developed for these problems can be applied. However, these algorithms are far from being efficient in terms of computational time involved when solving large problems. Hence metaheuristic approaches have been applied for solving the LBP problem, like genetic algorithms, which are the topic of this paper, simulated annealing [1,18] or tabu search [8,16,19].

Genetic algorithms are stochastic search techniques inspired by natural biological evolution. They were introduced and developed by Holland [12]. Since then, they have been implemented in a wide variety of applications and have become increasingly popular as a means of finding good solutions to hard optimization problems in acceptable computational times [9,10,14,17]. In a genetic algorithm, each solution to the problem at hand is encoded as a string of symbols which is called a chromosome. Each position of a symbol in the chromosome is called a 'gene' and its value is called the 'allele value'. To start, an initial population of chromosomes is generated and the fitness of each chromosome, which measures its quality, is assessed. Through successive iterations of the algorithm, called generations, the genetic algorithm maintains a population of chromosomes. To produce each new generation, offspring are formed by modifying a chromosome using a mutation operation or by combining chromosomes from the current generation (parents) using a crossover operation. Once the fitness of the resulting chromosomes is assessed, a selecting operation allows some of the parents and offspring to survive to the next generation and rejects the others, in general maintaining the same population size.

Concerning LBP problems, Mathieu et al. [13] developed a procedure in which chromosomes are $n_1 + n_2$ strings of base-10 digits that represent feasible solutions, not necessarily extreme points. The first level variables are generated while the second level variables are obtained by solving the second level linear programming problem (3). In this procedure, only mutation is used. Nishizaki et al. [15] firstly transform the LBP problem into a nonlinear one-level problem by including KKT conditions of the second level problem as constraints of the first level problem. Next, by introducing 0–1 variables with respect to the complementary conditions contained in the KKT conditions, this problem is transformed into a one-level mixed 0–1 programming problem. Then, a genetic algorithm is developed in which chromosomes are $m + n_2$ binary strings that represent bilevel feasible solutions. Crossover and mutation are applied and heuristic rules are given to improve the performance of the procedure. Hejazi et al. [11] also replace the second level problem by its KKT conditions. Then, they propose a genetic algorithm in which each chromosome is a string representing a bilevel feasible extreme point, that consists of $m + n_2$ binary components. Taking into account multiplicative constraints of the new problem, to check the feasibility of a chromosome is equivalent to solving two linear problems. Moreover, bearing in mind the feasible regions of these problems, they propose rules to discard chromosomes which are not feasible. Mutation and crossover are done in the usual way. Experimental results show the efficiency of this algorithm both from the computational point of view and the quality of solutions.

3. GABB: Genetic algorithm based on bases

As mentioned above, there is an extreme point of the polyhedron S which solves the LBP problem. Therefore, a clever examination of its extreme points provides an algorithm that will find the solution to the LBP problem in a finite number of steps. This is unsatisfactory, however, since the number of extreme points of S is, in general, very large. Therefore, we propose to combine the underlying structure of genetic algorithms with some kind of extreme point enumeration to develop an algorithm which provides optimal or near-optimal solutions in reasonable computational time. The main idea underlying the genetic algorithm developed is to associate chromosomes with extreme points of S. The fitness of a chromosome will evaluate its quality, penalizing the fact that the associated extreme point is not in IR. After constructing the first population of chromosomes, the genetic algorithm proceeds by performing crossover, mutation, evaluation and selection, until the stopping condition is met. In each generation a population of extreme points of S is maintained. A sketch of the algorithm is shown in Fig. 1. Next we describe in more detail the steps of the algorithm.

```
Genetic Algorithm Based on Bases
begin
    Set control parameters: Population size, $ps$, crossover probability, $p_c$,
        mutation probability, $p_m$ and stopping condition.
    Select a $ps$-sized population $Q$ of extreme points of $S$.
    Identify the population $P$ of chromosomes associated with $Q$.
    While the stopping condition is not verified:
        Select chromosomes from $P$ for crossover with probability $p_c$.
        Perform crossover.
        Merge $P$ and the offspring which are extreme points of $S$.
        Select chromosomes from $P$ for mutation with probability $p_m$.
        Perform mutation.
        Merge $P$ and the new chromosomes.
        Evaluate fitness of chromosomes in $P$.
        Sort out all chromosomes in $P$ in descending order of fitness value.
        Remove from $P$ all except the best $ps$ chromosomes.
end
```

Fig. 1. Genetic algorithm to solve the LBP problem.

3.1. Chromosome encoding

After introducing the vector $u \in \mathbb{R}^m$ of slack variables, which are controlled by the second level decision-maker, the polyhedron S can be written as

$$\begin{aligned} &A_1x_1 + A_2x_2 + u = b, \\ &x_1, x_2, u \geqslant 0. \end{aligned} \tag{4}$$

In order to associate chromosomes with extreme points of the polyhedron S, that is to say basic feasible solutions of (4), we encode each chromosome as an m string of integers whose components are the indices of the basic variables.

Given a chromosome C, we will denote by I_1^C and I_2^C the sets of indices of the first level variables and the second level ones, respectively. Notice that chromosomes do not necessarily represent bilevel feasible solutions. Abusing notation, each chromosome that represents a bilevel feasible solution will be called a feasible chromosome. If not, it will be a nonfeasible chromosome.

3.2. Initial population

The initial population is formed by ps chromosomes associated with basic feasible solutions of (4). Hence any general procedure to get them can be applied. Nevertheless, in the implementation we have preferred to build the initial population with feasible chromosomes. This device guarantees that, upon termination, the method will provide a bilevel feasible solution. Hence we suggest getting the initial population by solving ps times the linear programming problem:

$$\begin{aligned} \max_{x_1,x_2} \quad & d_1x_1 + d_2x_2 \\ \text{s.t.} \quad & A_1x_1 + A_2x_2 + u = b, \\ & x_1, x_2 \geqslant 0, \end{aligned} \tag{5}$$

where d_1 is an n_1-dimensional random row vector which is generated ps times.

3.3. Crossover

Crossover is the process by which chromosomes selected from a source population are combined to form offspring which are potential members of a successor population. In GABB, each chromosome of the current generation is selected for the crossover operation with probability p_c. Then, the parents selected are taken in pairs in order to provide offspring.

We propose two types of crossover operations. The first one, called 'variable-to-variable' (v-t-v) crossover, is more classical since it actually exchanges indices of basic variables between the two chromosomes. The general idea is shown in the following example:

Let us assume that $m = 8$. Let the two parents selected for crossover be

$$\begin{aligned} &\text{Parent 1:} \quad (2\quad \mathbf{4}\quad 5\quad 8\quad 9\quad 12\quad 21\quad \mathbf{22}) \\ &\text{Parent 2:} \quad (\mathbf{4}\quad 6\quad 13\quad 17\quad \mathbf{22}\quad 24\quad 25\quad 28) \end{aligned}$$

First, shared indices (in bold) are eliminated and a location is randomly selected (the third one in this example):

$$\begin{array}{cc|cccc} 2 & 5 & 8 & 9 & 12 & 21 \\ & & & & & \\ 6 & 13 & 17 & 24 & 25 & 28 \end{array}$$

Secondly, the right-hand indices are exchanged in pairs:

$$\begin{array}{ccccccccccccc} 2 & 5 & \mathbf{8} & \mathbf{9} & \mathbf{12} & \mathbf{21} & & 2 & 5 & 17 & 24 & 25 & 28 \\ & & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \rightsquigarrow & & & & & & \\ 6 & 13 & \mathbf{17} & \mathbf{24} & \mathbf{25} & \mathbf{28} & & 6 & 13 & 8 & 9 & 12 & 21 \end{array}$$

Finally, offspring are obtained by adding on to them 4 and 22, which are the shared indices previously eliminated:

$$\begin{array}{ll} \text{Offspring 1:} & (2 \quad 4 \quad 5 \quad 17 \quad 22 \quad 24 \quad 25 \quad 28) \\ \text{Offspring 2:} & (4 \quad 6 \quad 8 \quad 9 \quad 12 \quad 13 \quad 21 \quad 22) \end{array}$$

The major drawback of this crossover operation is the possibility of getting chromosomes associated with solutions not in S, which have to be rejected since they are not acceptable. To avoid this happening we propose a second crossover operation, called 'basis-to-basis' (b-t-b) crossover. It coincides with the previous one up to the moment in which the exchange location is randomly selected. Then, one by one, in ascending order of indices, the variables having the right-hand indices of the second parent enter the basis associated with the first parent. The minimum ratio test rule of simplex algorithm is applied in order to determine the variable leaving the basis. A similar process is followed with the variables having the right-hand indices of the first parent entering the basis associated with the second one. As a result, the new chromosomes are associated with solutions in S. Nevertheless, we cannot ensure that there is an exchange of indices in the classical sense, nor that all the selected indices will remain in the new chromosome at the end of the process.

After crossover, the accepted offspring are added to the current population and the whole set of chromosomes is passed on to the mutation step.

3.4. Mutation

Mutation is an operator which acts on an individual chromosome by exchanging one of its indices. In GABB, each chromosome is selected for the mutation operation with probability p_m. According to the mutation process, once a chromosome C has been chosen, a variable is randomly selected from the set of variables whose indices are not in C. Then this variable enters the basis associated with C and the minimum ratio test rule of simplex algorithm is applied to determine the variable leaving the basis. The new chromosome contains the indices of the new basis. Notice that the mutation operation results in a chromosome associated with a solution in S.

After mutation, the evaluation and selection processes are carried out on the current set of chromosomes available (current population plus offspring resulting from the crossover and mutation operations).

3.5. Fitness evaluation

The fitness function evaluates the quality of the chromosome. On the one hand, since we are looking for feasible chromosomes it assigns a better value to them. On the other hand, since we are looking for an optimal solution to the LBP problem it takes into account the first level objective function. Specifically, we define the fitness of a chromosome as the value of the first level objective function of the corresponding basic feasible solution corrected by a penalty term if the chromosome is nonfeasible.

Let C be a chromosome and (x_1, x_2, u) be the corresponding basic feasible solution. Then, the fitness value of C is defined as

$$f(C) = c_1x_1 + c_2x_2 - M(1 - \delta_{\mathrm{IR}}), \tag{6}$$

where M is a large constant and δ_{IR} denotes the indicator function for the set IR.

In order to check that C is a feasible chromosome, we have to prove that (x_2, u) solves problem (3). Clearly it is a feasible solution, hence we have only to prove the optimality. The dual problem of problem (3) is

$$\begin{aligned} \min_{w} \quad & w(b - A_1x_1) \\ \text{s.t.} \quad & wA_2 - v = d_2, \\ & w, v \geqslant 0, \end{aligned} \tag{7}$$

where w is an m-dimensional row vector of dual variables and v is an n_2-dimensional row vector of slack variables. As a result of the fundamental theorem of duality in linear programming, since problem (3) possesses an optimal solution, then problem (7) possesses an optimal solution and the two optimal objective values are equal. Besides, it is well known that (x_2, u), a feasible solution of (3), solves this problem if and only if there exists (w, v), a feasible solution of (7), such that

$$\begin{aligned} wu &= 0, \\ vx_2 &= 0. \end{aligned} \tag{8}$$

Moreover, the basic dual variables are complementary (their product is zero) to the nonbasic primal variables, and correspondingly, the nonbasic dual variables are complementary to the basic primal variables. Hence, by applying these conditions to the basic variable indices of set I_2^C, the corresponding variables in vectors w and v are nonbasic, i.e. they are zero. As a result, C is feasible if the following linear system has a solution

$$\begin{aligned} &wA_2 - v = d_2, \\ &w^j = 0, \quad j \in J, \quad v^k = 0, \ k \in K, \quad w, v \geqslant 0, \end{aligned} \tag{9}$$

where w^j refers to the jth component of vector w, v^k to the kth component of vector v, J is the index set of variables complementary to variables of u whose indices are in I_2^C and, similarly, K is the index set of variables complementary to variables of x_2 whose indices are in I_2^C. Notice that the problem of checking the bilevel feasibility of a solution has turned into a search for a feasible solution of system (9), with n_2 constraints and $m + n_2 - \text{card}(I_2^C)$ variables. This can be done, for instance, by applying the first phase of the two-phase method of linear programming.

3.6. Selection

The selection method of a genetic algorithm is designed to use fitness to guide the evolution of chromosomes. We use the elitist strategy which keeps the best chromosomes from one generation to the next.

3.7. Improvements

One of the main concerns of the algorithm is to solve problem (9) in order to check if a chromosome is feasible. To improve this process we can apply the following results which exploit the structure of the problem.

Lemma 1. *Two chromosomes which only differ in the genes associated to indices of the first level variables are simultaneously feasible or nonfeasible.*

Proof. Firstly, note that the feasible region of problem (7) does not depend on x_1, therefore dual problems associated to all $x_1 \in S_1$ have the same feasible region. Secondly, let C_1 and C_2 be two chromosomes which share the same second level basic variable indices, i.e. $I_2^{C_1} = I_2^{C_2}$. In order to check their feasibility we have to look for a feasible solution to the same system (9). Hence, the chromosomes are both feasible or nonfeasible. □

Theorem 2. *Let C_1 be a nonfeasible chromosome. If C_2 is a chromosome such that $I_2^{C_1} \subseteq I_2^{C_2}$, then C_2 is nonfeasible.*

Proof. Since C_1 is nonfeasible, the corresponding system (9) has no solution:

$$\begin{aligned} &wA_2 - v = d_2, \\ &w^j = 0, \quad j \in J_1, \quad v^k = 0, \ k \in K_1, \quad w, v \geqslant 0. \end{aligned} \tag{10}$$

Let us suppose that C_2 is feasible. Then the following system has a solution:

$$\begin{aligned} &wA_2 - v = d_2, \\ &w^j = 0, \quad j \in J_2, \quad v^k = 0, \ k \in K_2, \quad w, v \geqslant 0. \end{aligned} \tag{11}$$

Since $I_2^{C_1} \subseteq I_2^{C_2}$, then $J_1 \subseteq J_2$ and $K_1 \subseteq K_2$, i.e. system (11) has at least the same variables equal to zero as system (10). Hence any solution to system (11) solves system (10). Contradiction □

Theorem 3. *Let C_1 be a feasible chromosome. If C_2 is a chromosome such that $I_2^{C_2} \subseteq I_2^{C_1}$, then C_2 is feasible.*

Proof. Since C_1 is feasible, the following system has a solution:

$$\begin{aligned} &wA_2 - v = d_2, \\ &w^j = 0, \quad j \in J_1, \quad v^k = 0, \ k \in K_1, \quad w, v \geqslant 0. \end{aligned}$$

In the same way as before, taking into account that $J_2 \subseteq J_1$ and $K_2 \subseteq K_1$, this solution also solves the following system:

$$\begin{aligned} &wA_2 - v = d_2, \\ &w^j = 0, \quad j \in J_2, \quad v^k = 0, \ k \in K_2, \quad w, v \geqslant 0. \end{aligned}$$

Hence, C_2 is feasible. □

Theorem 4. *Let C_1 be a feasible chromosome. Let C_2 be the chromosome resulting from the mutation process. If the entering index corresponds to a first level variable, then C_2 is feasible.*

Proof. We distinguish two cases. If the leaving index also corresponds to a first level variable then, by applying Lemma 1, C_2 is feasible.

If the leaving index corresponds to a second level variable then $I_2^{C_2} \subset I_2^{C_1}$. By applying Theorem 3, C_2 is feasible. □

Theorem 5. *Let C_1 be a nonfeasible chromosome. Let C_2 be the chromosome resulting from the mutation process. If the leaving variable corresponds to a first level variable then C_2 is nonfeasible.*

Proof. If the entering index corresponds to a first level variable then the assertion is a consequence of Lemma 1. If not, it is sufficient to note that $I_2^{C_1} \subset I_2^{C_2}$ and apply Theorem 2. □

4. Computational results

A computational experiment was conducted to analyze the performance of GABB. Since it is a metaheuristic algorithm, with this experiment we aim to show the efficiency of our method in terms of the quality of the solution, measured through the first objective function value and the computational time involved. The computational study was divided into two parts. The first part is devoted to studying the effect of different parameters of the algorithm (crossover procedure, mutation probability, crossover probability and population size) in the quality of the solution provided by GABB. The purpose of this part of the study is to select the most efficient configuration for the algorithm. Once this configuration has been chosen, the second part of the study goes on to measure how good the obtained solutions are. For this purpose, first we compare GABB with the *K*th-best algorithm [4], which provides an exact optimal solution to LBP problems. Secondly, we compare GABB with the Hejazi et al. genetic algorithm [11], which has been proved to be efficient enough for these kind of problems.

The experiments were performed on a PC Pentium 4 at 2.6 GHz having 512 MB of RAM under Mandrake Linux 10.1. The genetic algorithm codes were written in MATLAB 6.1 Release 12.1. The LBP problems were randomly generated using the MATLAB environment. Coefficients of both level objective functions are randomly chosen from the uniform distribution on $(-10, 10)$. In order to assure that the polyhedron is bounded the coefficients of one constraint are generated from a uniform distribution on $(0, 10)$. The remaining elements of the coefficient matrix are generated from a uniform distribution on $(-10, 10)$. The right-hand side of each constraint is obtained as the sum of the absolute value of the coefficients in the constraint.

To get the initial population, the vector d_1 in problem (5) is taken as $d_1 = du$, where u is a n_1-dimensional random row vector whose components are generated according to a uniform distribution on $(-1, 1)$ and d is a constant defined as $d = \frac{1}{n_1}\sum_{j=1}^{n_1} \mid c_1^j \mid$, where c_1^j refers to the jth component of vector c_1. Moreover, in order to make use of Theorem 2, a *ps*-sized set NF of nonfeasible chromosomes is maintained. When a chromosome C is classified as nonfeasible, we directly add it to NF if room still exists. Otherwise, C replaces a chromosome $\tilde{C} \in$ NF such that $I_2^C \subset I_2^{\tilde{C}}$, if any exists.

A variety of test problems was used, which differ in the number of variables of the first and the second levels and in the number of constraints. There are three main problem groups defined by the number n of variables of the bilevel problem, excluding the slack variables, G_1: $n = 40$, G_2: $n = 60$ and G_3: $n = 100$. For each problem group we build test problem types having 30%, 50% and 80% of n as the number of second level variables n_2 and number of constraints m with the same percentages. Each combination of n_1, n_2 and m defines a test problem type. So, we have nine of these problem types for each problem group. Table 1 displays the problem dimensions. Note that the actual number of variables entailed in each problem is $n_1 + n_2 + m$, due

Table 1
Test problem dimensions

G_1: $n = 40$			G_2: $n = 60$			G_3: $n = 100$		
n_1	n_2	m	n_1	n_2	m	n_1	n_2	m
28	12	12	42	18	18	70	30	30
28	12	20	42	18	30	70	30	50
28	12	32	42	18	48	70	30	80
20	20	12	30	30	18	50	50	30
20	20	20	30	30	30	50	50	50
20	20	32	30	30	48	50	50	80
8	32	12	12	48	18	20	80	30
8	32	20	12	48	30	20	80	50
8	32	32	12	48	48	20	80	80

Table 2
Success or failure for each configuration of the algorithm

T_c	p_c	p_m	ps	$S(G_1)$	$F(G_1)$	$S(G_2)$	$F(G_2)$
v-t-v	0.25	0.25	50	218	52	177	93
v-t-v	0.25	0.25	100	235	35	196	74
v-t-v	0.25	0.5	50	217	53	180	90
v-t-v	0.25	0.5	100	237	33	205	65
v-t-v	0.5	0.25	50	240	30	205	65
v-t-v	0.5	0.25	100	252	18	218	52
v-t-v	0.5	0.5	50	240	30	205	65
v-t-v	0.5	0.5	100	247	23	219	51
b-t-b	0.25	0.25	50	231	39	204	66
b-t-b	0.25	0.25	100	250	20	215	55
b-t-b	0.25	0.5	50	233	37	206	64
b-t-b	0.25	0.5	100	250	20	213	57
b-t-b	0.5	0.25	50	242	28	207	63
b-t-b	0.5	0.25	100	248	22	229	41
b-t-b	0.5	0.5	50	238	32	207	63
b-t-b	0.5	0.5	100	251	19	225	45

$S(G_i)$: # of successes in problem group G_i.
$F(G_i)$: 270 − $S(G_i)$.

to the slack variables involved. Finally, we set the stopping condition at 200, 300 and 300 iterations for G_1, G_2 and G_3 problems, respectively.

4.1. Selecting the parameters of GABB

We study the influence of four factors in the performance of GABB and define a 2^4 factorial design that requires choosing two levels for each factor. This design allows us to analyze the impact of the four factors on each measure of the system's performance by carrying out 16 runs of the experiment. The factors and their levels are: Type of crossover operation (T_c = v-t-v or T_c = b-t-b); crossover probability ($p_c = 0.25$ or $p_c = 0.5$); mutation probability ($p_m = 0.25$ or $p_m = 0.5$); and population size ($ps = 50$ or $ps = 100$). As a result, each factor combination provides a configuration of the algorithm.

In this initial part of the experiment we use the problem groups G_1 and G_2. Within each problem type, 30 test problems are randomly generated. Hence, for each group, a total of $9 \times 30 = 270$ test problems are solved, each of them 16 times. For each test problem, let $f_1^{\max}$ be the best first level objective function value obtained from the 16 runs of the experiment. For each test problem and factor combination we define as a success, the first level objective function value of its solution being equal to $f_1^{\max}$. Table 2 shows the number of successes and failures obtained for each group and factor combination. For each configuration of the algorithm, we define the chances of success as the number of successes divided by 270. We take the chances of success as the response variable of the experimental design. The Analysis of Variance (ANOVA) table for group G_1 is displayed in Table 3. The type of crossover, the crossover probability and the population size are significant (the p-value being almost

Table 3
Analysis of variance for chances of success (problems G_1)

Estimated effects and coefficients for Psuccess (coded units)

Term	Effect	Coeff.	SE coeff.	T	P
Constant		0.88634	0.001934	458.31	0.000
Type of crossover	0.02639	0.01319	0.001934	6.82	0.001
Crossover probability	0.04028	0.02014	0.001934	10.41	0.000
Mutation probability	−0.00139	−0.00069	0.001934	−0.36	0.734
Population size	0.05139	0.02569	0.001934	13.29	0.000
Type of crossover * crossover probability	−0.02639	−0.01319	0.001934	−6.82	0.001
Type of crossover * mutation probability	0.00231	0.00116	0.001934	0.60	0.576
Type of crossover * population size	−0.00046	−0.00023	0.001934	−0.12	0.909
Crossover probability * mutation probability	−0.00417	−0.00208	0.001934	−1.08	0.331
Crossover probability * population size	−0.01620	−0.00810	0.001934	−4.19	0.009
Mutation probability * population size	0.00139	0.00069	0.001934	0.36	0.734

$S = 0.00773578$, R-Sq = 98.76%, R-Sq(adj.) = 96.27%
Analysis of variance for Psuccess (coded units)

Source	DF	Seq. SS	Adj. SS	Adj. MS	F	P
Main effects	4	0.0198457	0.0198457	0.00496142	82.91	0.000
2-Way interactions	6	0.0039352	0.0039352	0.00065586	10.96	0.009
Residual error	5	0.0002992	0.0002992	0.00005984		
Total	15	0.0240801				

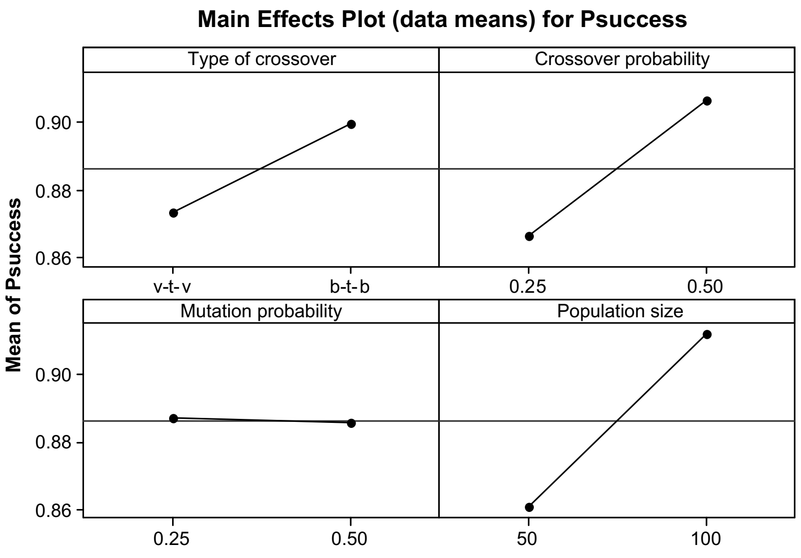

Fig. 2. Main factor plot (problem group G_1).

Table 4
The number of problems solved to optimality with the *K*th-best algorithm

Group G_1	# of problems	Group G_2	# of problems
n_1-n_2-m		n_1-n_2-m	
28-12-12	30	42-18-18	27
28-12-20	28	42-18-30	25
28-12-32	28	42-18-48	22
20-20-12	28	30-30-18	19
20-20-20	23	30-30-30	16
20-20-32	18	30-30-48	9
8-32-12	18	12-48-18	4
8-32-20	8	12-48-30	1
8-32-32	2	12-48-48	0

zero). The interactions between crossover probability and the type of crossover, and crossover probability and the population size are also significant. In order to illustrate graphically the influence of factors, Fig. 2 displays the main factor plot for the response variable in the same group. Similar results are obtained for problem group G_2. As a consequence, to get the best configuration for the algorithm the b-t-b crossover, $p_c = 0.5$ and $ps = 100$, should be selected, disregarding the selection $p_m = 0.25$ or 0.5. For what follows, we have opted for $p_m = 0.25$.

4.2. Assessing the performance of GABB

In order to get evidence of the quality of the results provided by GABB, with the selected parameter configuration, we start by comparing it with the

Table 5
The number of problems giving the best solution

Group G_1	GABB	GAH	Group G_2	GABB	GAH	Group G_3	GABB	GAH
n_1-n_2-m			n_1-n_2-m			n_1-n_2-m		
28-12-12	29	30	42-18-18	29	29	70-30-30	9	8
28-12-20	29	29	42-18-30	28	29	70-30-50	10	6
28-12-32	28	27	42-18-48	26	28	70-30-80	9	7
20-20-12	29	30	30-30-18	27	28	50-50-30	9	6
20-20-20	30	28	30-30-30	29	29	50-50-50	7	8
20-20-32	28	28	30-30-48	25	19	50-50-80	9	3
8-32-12	30	27	12-48-18	28	27	20-80-30	9	5
8-32-20	30	25	12-48-30	28	19	20-80-50	8	2
8-32-32	27	22	12-48-48	27	13	20-80-80	9	2

Groups G_1 and G_2 have 30 problems. Group G_3 has 10 problems.

Table 6
The average value of the iteration in which the best solution appears for the first time

Group G_1	GABB	GAH	Group G_2	GABB	GAH	Group G_3	GABB	GAH
n_1-n_2-m			n_1-n_2-m			n_1-n_2-m		
28-12-12	14.33	9.90	42-18-18	38.03	21.00	70-30-30	98.70	83.70
28-12-20	30.73	21.83	42-18-30	58.40	51.00	70-30-50	119.60	154.60
28-12-32	44.50	59.00	42-18-48	72.53	108.50	70-30-80	146.10	192.20
20-20-12	20.07	16.60	30-30-18	44.00	49.43	50-50-30	86.00	161.90
20-20-20	27.27	40.00	30-30-30	64.53	120.23	50-50-50	109.20	207.90
20-20-32	43.97	80.93	30-30-48	78.23	175.57	50-50-80	164.90	255.00
8-32-12	23.93	38.20	12-48-18	40.77	94.87	20-80-30	113.30	224.30
8-32-20	40.93	70.17	12-48-30	71.03	154.13	20-80-50	147.60	257.60
8-32-32	57.97	124.07	12-48-48	98.90	202.77	20-80-80	165.30	241.90
All	33.74	51.19	All	62.94	108.62	All	127.86	197.68

Kth-best algorithm [4]. As mentioned in previous sections, enumeration of all extreme points of the polyhedron S is an algorithm that finds an optimal solution to the LBP problem in a finite number of steps. This is unsatisfactory, however, since the number of extreme points of S is, in general, very large. The Kth-best algorithm provides a more successful enumeration scheme, although only small problems can be solved in reasonably short times. To make the paper self-contained, we include a brief description of the algorithm. First the relaxed problem (12) is solved

$$\begin{aligned} \max_{x_1,x_2} \quad & f_1(x_1,x_2) = c_1x_1 + c_2x_2 \\ \text{s.t.} \quad & A_1x_1 + A_2x_2 + u = b, \\ & x_1, x_2 \geqslant 0. \end{aligned} \tag{12}$$

Let $(x_1^{[1]}, x_2^{[1]})$ be an optimal solution. If this is a point of IR, then it is an optimal solution of the LBP problem. If this is not so, the set of its adjacent extreme points $W^{[1]}$ is considered. The algorithm maintains a set W of extreme points adjacent to extreme points previously computed. Initially, we make $W = W^{[1]}$ and the extreme point in W which provides the best value of f_1 is selected to test if it is a point of IR. If it is, the algorithm finishes. If this is not so, the point is eliminated from W and its adjacent extreme points with a worse value of f_1 are added to W. The algorithm continues by selecting the best extreme point in W with respect to f_1 and repeating the process. Upon termination, an extreme point of S belonging to IR with the best value of the f_1 is provided, so an optimal solution to the LBP problem.

By comparing GABB with the Kth-best algorithm we measure to what extent GABB provides optimal solutions to LBP problems. In this part of the experiment we only use problem groups G_1 and G_2, due to the rather large time involved in solv-

Table 7
The average value of ET, the earliest CPU time (in seconds) in which the best solution was obtained and average value of TT, the CPU total time involved in making all the iterations prescribed

Problems		Mean ET		Mean TT	
	n_1-n_2-m	GABB	GAH	GABB	GAH
G_1	28-12-12	3.01	5.21	28.55	97.69
	28-12-20	7.60	15.78	35.32	130.14
	28-12-32	12.75	60.04	42.73	190.37
	20-20-12	5.91	10.62	40.93	116.12
	20-20-20	10.07	36.89	50.85	170.79
	20-20-32	19.51	89.65	65.27	210.38
	8-32-12	10.70	38.36	74.21	182.26
	8-32-20	22.76	78.43	90.92	215.39
	8-32-32	39.21	169.01	111.36	267.27
	All	14.61	56.00	60.01	175.60
G_2	42-18-18	10.96	19.53	57.54	253.81
	42-18-30	19.06	60.76	70.02	340.69
	42-18-48	33.01	189.66	96.78	502.42
	30-30-18	22.11	61.46	103.19	357.53
	30-30-30	39.14	173.40	129.71	418.63
	30-30-48	64.45	339.80	176.27	562.22
	12-48-18	49.06	221.55	281.54	678.15
	12-48-30	109.29	417.52	363.51	789.42
	12-48-48	181.80	627.11	448.38	907.44
	All	58.77	234.53	191.88	534.48
G_3	70-30-30	53.52	144.36	117.18	491.88
	70-30-50	83.36	398.78	161.68	747.19
	70-30-80	143.34	1204.57	228.56	1849.67
	50-50-30	138.46	540.55	354.93	987.00
	50-50-50	225.81	862.63	455.87	1211.47
	50-50-80	414.54	1714.56	628.06	1998.15
	20-80-30	760.26	2816.91	1730.78	3675.80
	20-80-50	1392.21	3723.28	2387.26	4280.76
	20-80-80	1703.21	3822.20	2699.64	4712.09
	All	546.08	1691.98	973.77	2217.11

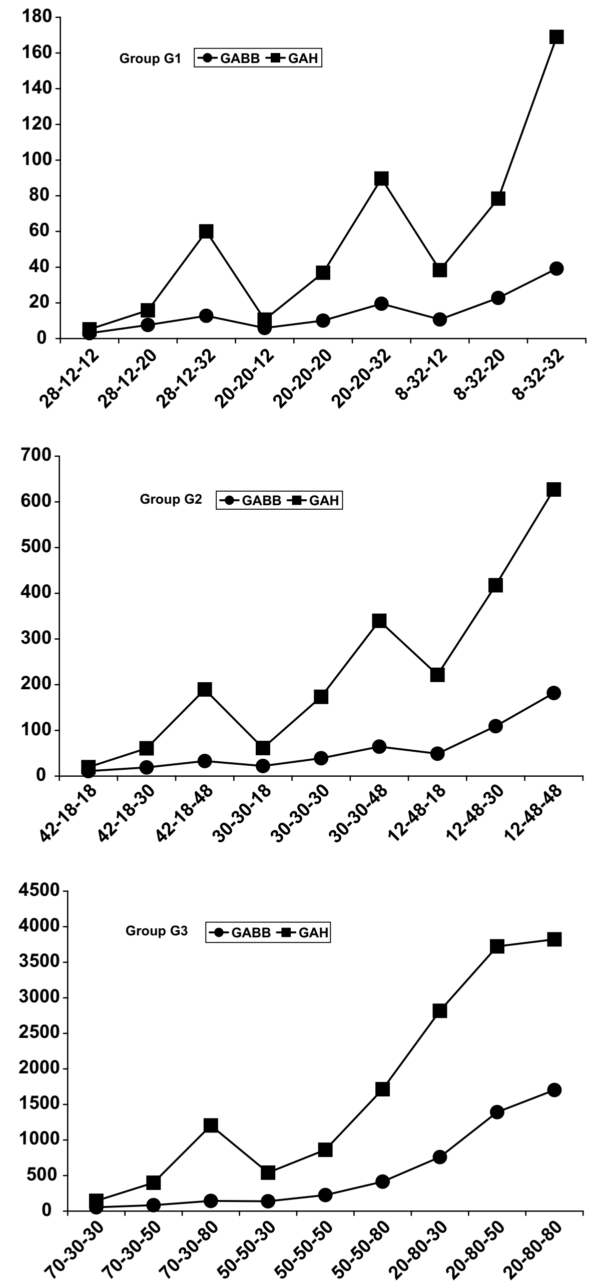

Fig. 3. The average of the earliest CPU time in which the best solution was obtained versus problem size.

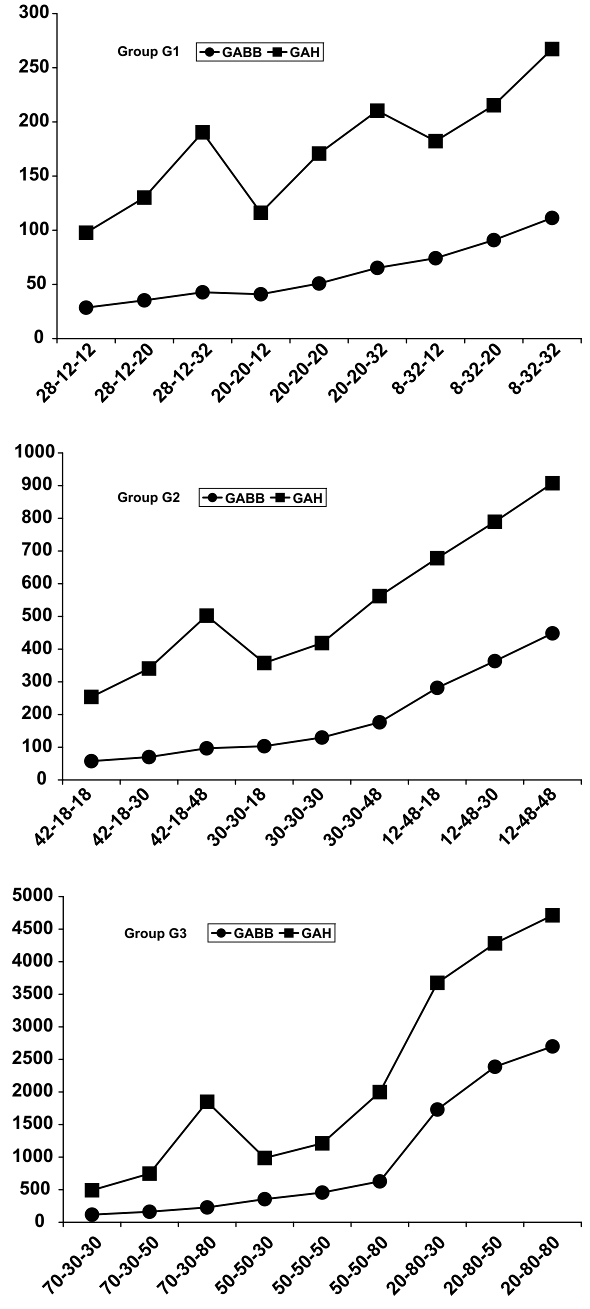

Fig. 4. The average of the CPU total time involved in making all the iterations prescribed versus problem size.

ing problems with the *K*th-best algorithm. Moreover, we decided to stop this latest algorithm after 1200 seconds of CPU time. Table 4 displays the number of problems for which the *K*th-best algorithm ended by giving an optimal solution. Notice that only 183 out of 270 problems of group G_1 and 123 out of 270 ones of group G_2 were solved in 1200 seconds of CPU time. It is noteworthy to point out that almost none of the largest problems were solved, so confirming the poor performance of the *K*th-best algorithm.

For these 306 problems, we compute the absolute value difference $|f_1^{K\text{best}} - f_1^{\text{GABB}}|$, where $f_1^{K\text{best}}$ and f_1^{GABB} stand for the first level objective function value of the solution provided by the respective

Table 8
Analysis of variance results for the earliest CPU time in which the best solution was obtained

Source	DF	Seq. SS	Adj. SS	Adj. MS	F	P
G_1						
A	1	7707.0	7707.0	7707.0	69.45	0.001
B	2	5763.5	5763.5	2881.7	25.97	0.005
C	2	8747.4	8747.4	4373.7	39.42	0.002
A * B	2	2140.4	2140.4	1070.2	9.64	0.030
A * C	2	4048.3	4048.3	2024.2	18.24	0.010
B * C	4	1206.7	1206.7	301.7	2.72	0.178
Error	4	443.9	443.9	111.0		
Total	17	30,057.2				
$S = 10.5340$, R-Sq = 98.52%, R-Sq(adj.) = 93.72%						
G_2						
A	1	139,024	139,024	139,024	239.20	0.000
B	2	143,179	143,179	71,589	123.18	0.000
C	2	92,999	92,999	46,500	80.01	0.001
A * B	2	44,629	44,629	22,315	38.39	0.002
A * C	2	36,424	36,424	18,212	31.33	0.004
B * C	4	15,497	15,497	3874	6.67	0.047
Error	4	2325	2325	581		
Total	17	474,076				
$S = 24.1081$, R-Sq = 99.51%, R-Sq(adj.) = 97.92%						
G_3						
A	1	5,908,939	5,908,939	5,908,939	132.02	0.000
B	2	14,368,051	14,368,051	7,184,026	160.51	0.000
C	2	1,724,163	1,724,163	862,082	19.26	0.009
A * B	2	2,417,977	2,417,977	1,208,988	27.01	0.005
A * C	2	316,580	316,580	158,290	3.54	0.130
B * C	4	248,571	248,571	62,143	1.39	0.379
Error	4	179,034	179,034	44,758		
Total	17	25,163,314				
$S = 211.562$, R-Sq = 99.29%, R-Sq(adj.) = 96.98%						

algorithm. Then, 290 problems out of 306 give the same solution, which is a remarkably close fit. In other words, in 94.77% of problems GABB provided the optimal solution.

The last part of the experiment is devoted to comparing GABB with the genetic algorithm proposed by Hejazi et al. (called GAH from now on), which has been briefly described in Section 2. The aim is to examine both algorithms in order to assess their differences in terms of the quality of the solution provided and the CPU time involved.

First, we compare GAH with the Kth-best algorithm, in the same way we did with GABB. Then, 288 problems out of 306 (i.e. 94.12%) give the same solution. So, for these problems, which are the smaller ones, GABB and GAH can be considered equivalent since both provide similar results in terms of achieving the optimal solution. Nevertheless, when analyzing the whole set of problems G_1 and G_2, GABB shows a better performance, which increases as the size of the problem gets larger, especially as the number of variables controlled by the second level increases. Table 5 presents these results. At a glance, the first noteworthy result is that the number of problems for which GABB provides the best solution does not seem to be affected very much by the number of variables or constraints, which is not the case with GAH.

For the 270 problems of group G_1, GABB gives the best solution in 260, i.e. a proportion of 96%, while GAH gives the best solution in 246, i.e. a proportion of 91%. The p-value of the test for the difference between two population proportions is 0.013, so both proportions are statistically different. The 95% confidence interval for the difference between the proportions of GABB and GAH is $(1.11, 9.25)$. Since it does not include zero, the proportion for GABB exceeds the proportion for GAH. In fact, we can state that we are 95% confident that the proportion of GABB exceeds that of GAH by between 1.11 and 9.25. Similarly, for the 270 problems of group G_2, these data are 247, i.e. 91%, for GABB and 221, i.e. 82%, for GAH. Again, the statistical test results indicate that there are significant

Table 9
Analysis of variance results for the CPU total time involved in making all the iterations prescribed

Source	DF	Seq. SS	Adj. SS	Adj. MS	*F*	*P*
G_1						
A	1	60,123.9	60,123.9	60,123.9	1078.35	0.000
B	2	15,152.8	15,152.8	7576.4	135.89	0.000
C	2	10,115.7	10,115.7	5057.8	90.71	0.000
A * B	2	502.4	502.4	251.2	4.51	0.095
A * C	2	3224.4	3224.4	1612.2	28.92	0.004
B * C	4	105.8	105.8	26.4	0.47	0.756
Error	4	223.0	223.0	55.8		
Total	17	89,448.0				
$S = 7.46696$, R-Sq = 99.75%, R-Sq(adj.) = 98.94%						
G_2						
A	1	528,177	528,177	528,177	732.11	0.000
B	2	430,755	430,755	215,377	298.54	0.000
C	2	78,205	78,205	39,103	54.20	0.001
A * B	2	16,369	16,369	8184	11.34	0.022
A * C	2	14,010	14,010	7005	9.71	0.029
B * C	4	2573	2573	643	0.89	0.543
Error	4	2886	2886	721		
Total	17	1072975				
$S = 26.8597$, R-Sq = 99.73%, R-Sq(adj.) = 98.86%						
G_3						
A	1	6,956,518	6,956,518	6,956,518	144.41	0.000
B	2	24,916,797	24,916,797	12,458,399	258.63	0.000
C	2	1,914,005	1,914,005	957,002	19.87	0.008
A * B	2	1,127,208	1,127,208	563,604	11.70	0.021
A * C	2	412,206	412,206	206,103	4.28	0.101
B * C	4	161,911	161,911	40,478	0.84	0.565
Error	4	192,687	192,687	48,172		
Total	17	35,681,332				
$S = 219.480$, R-Sq = 99.46%, R-Sq(adj.) = 97.70%						

differences between both proportions with a *p*-value equal to zero. The 95% confidence interval is now (3.95, 15.31).

In order to study more in depth this fact, we include at this moment of the study the group G_3. Bearing in mind the time taken when solving these problems, we decided to randomly generate only 10 test problems within each problem type in this group, instead of the 30 test problems randomly generated for groups G_1 and G_2. That is to say, 90 more problems were solved, each of them with both algorithms. The three last columns in Table 5 show the results. In this case, in 79 out of 90 problems, i.e. a proportion of 88%, GABB gives the best solution, whereas GAH only gives the best solution in 47 problems, i.e. a proportion of 52%. So, for this group we also reject that both proportions are equal at any level of significance (the *p*-value being zero). The 95% confidence interval for the difference between proportions is in this case (23.21, 47.90). Therefore, we can conclude from the experiment that there is strong evidence to support the claim that both methods are different. In fact, based on data we can assert that there is a statistically significant difference in proportions in favor of GABB, meaning that GABB gives better solutions. Furthermore, the ability of GAH to provide the best solution decreases when the complexity of the problem rises, whereas that of GABB remains steady.

We have also studied the iteration in which the best solution appears for the first time, EI. Table 6 displays the average value of EI in each problem type. The last row shows the average value for the whole group. By looking at these results, we see that GABB seems to be more steady in the sense that it is less affected by changes in the number of variables and constraints. Moreover, in almost every problem type it is possible to prescribe a smaller number of iterations to GABB than to GAH obtaining just as good results. Specifically, statistical analysis of data indicates that GAH needs, on the average, more iterations than does GABB (the p-value being zero). The 95% confidence intervals for the difference between the means of GAH and GABB are (12.50, 22.39), (37.43, 53.92), (55.44, 84.20), respectively for groups G_1, G_2 and G_3. So, for instance, for group G_2 we can assert that we are 95% confident that GAH needs between 37.43 and 53.92 more iterations than does GABB.

There are also remarkable differences between GABB and GAH when examining CPU times involved (in seconds). We have analyzed the earliest CPU time in which the best solution was obtained, ET, and the CPU total time involved in making all the iterations prescribed, TT. Table 7 presents the mean of ET and TT in each problem type. The last row of each group shows the average value for the whole group. It is clear that GABB always takes much less time to get the best solution, and this fact increases with the complexity of the problem. Similar comments can be made on the total time to finish the prescribed iterations, with even more impressive differences. Figs. 3 and 4 make the advantages of GABB against GAH even more evident.

In order to assess the actual effect of the algorithm on ET and TT, a factorial design was defined considering as factors: Factor A, the algorithm; factor B, the percentage of n as the number of second level variables n_2; factor C, the percentage of n as the number of constraints m; and factor D, the size n. As expected, the results indicated that the most influent factor was D. The effect was so important that the influence of the rest of factors was clouded. So, we decided to study the joint effect of factors A, B and C for each group. Tables 8 and 9 exhibit the results. For ET, the algorithm is the second most influent factor. It explains 25.6%, 29.3% and 23.5% of variability, respectively for groups G_1, G_2 and G_3. Similarly, for TT, the algorithm is the most influent factor for groups G_1 and G_2, explaining 67.2% and 49.2% of variability, respectively. For the group G_3, the algorithm is the second most influent factor and explains 19.5% of variability. Therefore, we can conclude that the algorithm really influences both CPU times. Moreover, the algorithm GABB is less time consuming.

5. Conclusions

In this paper a genetic algorithm for the LBP problem has been developed and implemented. Results of a variety of problems show that the algorithm provides an effective and useful solution approach to complex problems. Its main idea is to associate chromosomes with extreme points of the polyhedron defined by the common constraints at both levels. Moreover, by using the properties of LBP problems several rules are proposed which improve the process of checking the feasibility of chromosomes.

Finally, it is worth pointing out that the proposed algorithm relies on the existence of an extreme point of S which is an optimal solution of the LBP problem and the fact that the second level problem is a linear one. So, we can conclude that this algorithm can still be used to solve a wider class of bilevel problems. Indeed, Calvete and Galé [5] show that the quasiconcave bilevel problem, in which both level decision-makers minimize quasiconcave objective functions and the common constraint region is a polyhedron, also has an optimal solution which is an extreme point of the polyhedron. This model includes bilevel problems where the first level objective function is linear, fractional (ratios of concave nonnegative functions and convex strictly positive functions) or multiplicative (the product of a set of concave functions, each strictly positive). Therefore, GABB can be applied to solve a particular case of these problems in which the second level objective function is linear. Future work will focus on the development of genetic algorithms for more general bilevel problems.

References

[1] G. Anandalingam, R. Mathieu, C.L. Pittard, N. Sinha, Artificial intelligence based approaches for solving hierarchical optimization problems, in: R. Sharda, B.L. Golden, E. Wasil, O. Balci, W. Stewart (Eds.), Impacts of Recent Computer Advances on Operations Research, North-Holland, New York, 1989, pp. 289–301.
[2] J.F. Bard, Practical Bilevel Optimization. Algorithms and Applications, Kluwer Academic Publishers, Dordrecht, Boston, London, 1998.
[3] W.F. Bialas, M.H. Karwan, On two-level optimization, IEEE Transactions on Automatic Control 27 (1982) 211–214.
[4] W.F. Bialas, M.H. Karwan, Two-level linear programming, Management Science 30 (1984) 1004–1024.
[5] H.I. Calvete, C. Galé, On the quasiconcave bilevel programming problem, Journal of Optimization Theory and Applications 98 (3) (1998) 613–622.
[6] S. Dempe, Foundations of Bilevel Programming, Kluwer Academic Publishers, Dordrecht, Boston, London, 2002.
[7] S. Dempe, Annotated bibliography on bilevel programming and mathematical programs with equilibrium constraints, Optimization 52 (2003) 333–359.

[8] M. Gendreau, P. Marcotte, G. Savard, A hybrid tabu-ascent algorithm for the linear bilevel programming problem, Journal of Global Optimization 9 (1996) 1–14.

[9] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA, 1989.

[10] D.E. Goldberg, The Design of Innovation: Lessons from and for Competent Genetic Algorithms, Kluwer Academic Publishers, Boston, 2002.

[11] S.R. Hejazi, A. Memariani, G. Jahanshahloo, M.M. Sepehri, Linear bilevel programming solution by genetic algorithm, Computers and Operations Research 29 (2002) 1913–1925.

[12] J.H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI, 1975.

[13] R. Mathieu, L. Pittard, G. Anandalingam, Genetic algorithms based approach to bi-level linear programming, RAIRO-Operations Research 28 (1) (1994) 1–22.

[14] Z. Michalewick, Genetic Algorithms + Data Structures = Evolution Programs, third ed., Springer, Berlin, 1996.

[15] I. Nishizaki, M. Sakawa, K. Niwa, Y. Kitaguchi, A computational method using genetic algorithms for obtaining Stackelberg solutions to two-level linear programming problems, Electronics and Communications in Japan, Part 3 85 (6) (2002) 55–62.

[16] J. Rajesh, K. Gupta, H.S. Kusumakar, V.K. Jayaraman, B.D. Kulkarni, A tabu search based approach for solving a class of bilevel programming problems in chemical engineering, Journal of Heuristics 9 (2003) 307–319.

[17] C.R. Reeves, Genetic algorithms for the operations researcher, INFORMS Journal on Computing 9 (3) (1997) 231–250.

[18] K.H. Sakin, A.R. Ciric, A dual temperature simulated annealing approach for solving bilevel programming problems, Computers and Chemical Engineering 23 (1998) 11–25.

[19] U.P. Wen, A.D. Huang, A simple tabu search method to solve the mixed-integer linear bilevel programming problem, European Journal of Operational Research 88 (3) (1996) 563–571.