J. Appl. Math. & Informatics Vol. **28***(2010), No. 3 - 4, pp. 597 - 610*
Website: http://www.kcam.biz

A GENETIC ALGORITHM BASED ON OPTIMALITY CONDITIONS FOR NONLINEAR BILEVEL PROGRAMMING PROBLEMS

HECHENG LI* AND YUPING WANG

ABSTRACT. For a class of nonlinear bilevel programming problems in which the follower's problem is linear, the paper develops a genetic algorithm based on the optimality conditions of linear programming. At first, we denote an individual by selecting a base of the follower's linear programming, and use the optimality conditions given in the simplex method to denote the follower's solution functions. Then, the follower's problem and variables are replaced by these optimality conditions and the solution functions, which makes the original bilevel programming become a single-level one only including the leader's variables. At last, the single-level problem is solved by using some classical optimization techniques, and its objective value is regarded as the fitness of the individual. The numerical results illustrate that the proposed algorithm is efficient and stable.

AMS Mathematics Subject Classification : 90C26, 90C30, 90C46, 90C59
Key words and phrases : Nonlinear bilevel programming; genetic algorithm; optimality conditions; optimal solutions

1. Introduction

The bilevel programming problem (BLPP) can be viewed as a static version of the noncooperative, two-person game introduced by Von Stackelberg in the context of unbalanced economic markets[14], as a result, it is also known as Stackelberg problem. This kind of problems involve two optimization problems at different levels, in which the feasible region of one optimization problem (leader's problem/upper level problem) is implicitly determined by the other (follower's problem/lower level problem). The general bilevel programming problem can be formulated as follows

Received August 13, 2009. Revised August 21, 2009. Accepted September 25, 2009.
*Corresponding author.

$$\begin{cases} \min\limits_{x \in X} F(x, y) \\ s.t.\ G(x, y) \leq 0 \\ \min\limits_{y \in Y} f(x, y) \\ s.t.\ g(x, y) \leq 0 \end{cases} \tag{1}$$

where $x \in R^n, y \in R^m$; The variables of problem (1) are divided into two classes, namely the leader's variables x and the follower's variables y. Similarly, $F(f) : R^n \times R^m \to R$ is called the leader's (follower's) objective function, while the vector-valued functions $G : R^n \times R^m \to R^p$ and $g : R^n \times R^m \to R^q$ are called the leader's and follower's constraints respectively. The sets X and Y place additional constraints on the variables, such as upper and lower bounds or integrality requirements etc. This mathematical programming model arises when two independent decision makers, ordered within a hierarchical structure, have conflicting objectives, and each decision maker seeks to optimize his/her objective function. In model(1), The leader moves first by choosing a vector $x \in X \subseteq R^n$ in an attempt to optimize his/her objective function $F(x, y)$; the leader's choice of strategies affects both the follower's objective and decision space. The follower observes the leader's choice and reacts by selecting a vector $y \in Y \subseteq R^m$ that optimize his/her objective function $f(x, y)$. In doing so, the follower affects the leader's outcome.

BLPP is widely used to lots of fields such as economy, control, engineering and management[2, 5], and more and more practical problems can be formulated as the bilevel programming models, so it is important to design all types of effective and efficient algorithms to solve different types of BLPPs. But due to its nested structure, BLPP is intrinsically difficult, it has been reported that BLPP is strongly NP-hard[2]. When all functions involved are linear, BLPP is called linear bilevel programming. It is the simplest one among the family of BLPPs, and the optimal solutions can occurs at vertices of feasible region. Based on these properties, lots of algorithms are proposed to solve this kind of problems[2, 3, 7, 17]. When the functions involve nonlinear terms, the corresponding problem is called nonlinear BLPP, which is more complex and challenging than linear one. To date, some exact algorithms have been developed for some special cases of nonlinear bilevel programs[1, 2, 4, 5, 6, 8, 10, 13, 15, 16, 19], and some intelligent algorithms, such as evolutionary algorithms(EAs)/genetic algorithms(GAs)[9, 11, 16, 18], tuba search approaches[12], etc, have been applied to obtain the optimal solutions of nonlinear BLPPs. However, Amongst the algorithms, some involve the application of enumerative methods, while others replace the follower's problem with its K-K-T conditions or apply penalty function methods. But all of them are very time consuming, especially when the follower's problem is large. In order to overcome the disadvantage, the paper is devoted to designing an algorithm which can reduce the computational complexity caused by the follower's solution.

In this paper we consider a special class of nonlinear BLPP, in which the leader's functions are convex in all variables, and the follower's programming is linear. The paper develops a genetic algorithm based on the optimality conditions of linear programming(LP). The algorithm aims to combine an extreme point enumeration technique in the follower's problem with the optimization of the leader's problem. At first, we denote an individual by selecting a base of the follower's linear programming, and use the optimality conditions given in the simplex method to denote the follower's solutions which are functions of leader's variables. Then, the follower's problem and variables are replaced by these optimality conditions and solution functions, which makes the original BLPP become a single-level convex programming only including the leader's variables. At last, the single-level problem is solved by using some classical optimization techniques, and its objective value is regarded as the fitness of the individual.

This paper is organized as follows. The model of the bilevel programming problem and some notations are presented in Section 2, and a genetic algorithm is given based on the optimality conditions in Section 3. Experimental results and comparison are presented in Section 4. We finally conclude our paper in Section 5.

2. Discussed model and basic notations

Let us consider the bilevel programming problem defined as follows

$$\begin{cases} \min\limits_{x\in X} F(x,y) \\ s.t.\ G(x,y)\le 0 \\ \min\limits_{y\in Y} f(x,y)=\bar{C}(x)y \\ s.t.\ Ux+Vy+W\le 0, y\ge 0. \end{cases} \tag{2}$$

where F, G are convex and twice continuously differential in all variables (x, y), the vector-valued function $\bar{C}(x) = (c_1(x), c_2(x), \ldots, c_m(x))$, here $c_i(x), i = 1, 2, \ldots, m$, are linear in x. U is a $q \times n$-matrix, V is a $q \times m$-matrix, and $W \in R^q$. X and Y are box sets as follows:

$$X = \{(x_1, x_2, \ldots, x_n)^T \in R^n \mid x_i \in [l_i, u_i], i = 1, 2, \ldots, n\}$$

$$Y = \{(y_1, y_2, \ldots, y_m)^T \in R^m \mid y_j \in [\bar{l}_j, \bar{u}_j], j = 1, 2, \ldots, m\}$$

where $l_i, u_i, \bar{l}_j, \bar{u}_j$ are all real constants.

Now we introduce some related definitions and notations[2] as follows.

1) Search space: $\Omega = \{(x, y)|x \in X, y \in Y\}$.

2) Constraint region: $S = \{(x, y) \in \Omega|G(x, y) \le 0, Ux + Vy + W \le 0, y \ge 0\}$.

3) Feasible region of follower's problem for x fixed: $S(x) = \{y \in Y|Ux+Vy+W \le 0, y \ge 0\}$.

4) Projection of S onto the leader's decision space: $S(X) = \{x \in X|\exists y, (x, y) \in S\}$.

5) Follower's rational reaction set for each $x \in S(X)$: $M(x) = \{y \in Y | y \in argmin\{\bar{C}(x)v, v \in S(x)\}\}$.

6) Inducible region: $IR = \{(x, y) \in S | y \in M(x)\}$.

In terms of aforementioned definitions, problem (2) can also be written as:

$$\min\{F(x, y) | (x, y) \in IR\}$$

In order to ensure that problem (2) is well posed, in the remainder, we always assume that

A1: S is nonempty and compact.

A2: For all decisions taken by the leader, each follower has some room to react, that is, $S(x) \neq \phi$.

A3: The follower's problem has unique optimal solution for each fixed x.

A4: The rank of matrix V is q.

Since $y \in Y$ can be written as linear constraints, for the purpose of simplicity, we always omit the additional restrict in the remainder sections.

3. Solution method

For nonlinear BLPP, the existing algorithms can be divided into two classes. One always begins with leader's variables, at first, an $x \in X$ is selected, then for the fixed x, follower's problem is solved to obtain y. In this process, the leader always tries to optimize his/her objective value by searching x. While the other uses some techniques to transform BLPP into a single level programming, such as penalty functions or K-K-T conditions. When the follower's programming is a large-scale problem, these algorithms are time-consuming. We try to solve the problem from another angle. At first, we encode each individual by using the potential base of follower's programming, that is, a base corresponds to an individual. When the base is optimal and feasible for some x, the optimality conditions and the follower's solutions can be denoted by the linear inequalities and functions of x, respectively. Then the follower's programming can be replaced by these optimality conditions, and the y in the leader's problem can also be replaced by the solution functions. As a result, the original BLPP can be transformed into a single level programming without follower's variables, which is different from the single level programming obtained by using K-K-T conditions or penalty function methods, since the two techniques can't eliminate follower's variables. Next we describe in more details the steps of the algorithm.

3.1. Chromosome encoding

The follower's problem of (2) can be re-written as:

$$\begin{cases} \min\limits_{z} f(x, z) = C(x)z \\ s.t.\ Az = b(x), z \geq 0. \end{cases} \tag{3}$$

where $A = (V, I), b(x) = -W - Ux$, $z = (y^T, y_0^T)^T \in R^{m+q}$, $C(x) = (\bar{C}(x), \mathbf{0})$, and y_0 is a slack vector.

Arbitrarily select a base B of problem (3), we denote the indices of basic variables by $i_1, i_2, \ldots, i_q$ and set $i_1 < i_2 < \cdots < i_q$. Then an individual can be represented by $(0, \ldots, 0, 1, 0, \ldots, 0, \ldots, 0, 1, 0, \ldots, 0)$, i.e. the i_jth component of the individual is 1, $j = 1, \ldots, q$, while others are 0.

3.2. Fitness evaluation

For each individual $l = (0, \ldots, 0, 1, 0, \ldots, 0, \ldots, 0, 1, 0, \ldots, 0)$, the corresponding basic matrix is denoted by B. According to the theory of the simplex method, if there exists at least an $x \in X$ such that the inequalities

$$\begin{cases} B^{-1}b(x) \geq 0 \\ C(x) - C_B(x)B^{-1}A \geq 0 \end{cases} \tag{4}$$

hold, where $C_B(x)$ are the components of $C(x)$ corresponding to basic variables, then for each x satisfying (4), $B^{-1}b(x)$ provides an optimal solution $z(x) = (y(x), y_0(x))$ to the follower's problem, in which the basic variable values of the solution are taken as $B^{-1}b(x)$, while the values of nonbasic variables are 0. As a result, a single level programming is got as follows

$$\begin{cases} \min\limits_{x \in X} F(x, y(x)) \\ s.t.\ G(x, y(x)) \leq 0 \\ B^{-1}b(x) \geq 0 \\ C(x) - C_B(x)B^{-1}A \geq 0 \end{cases} \tag{5}$$

It is easily seen that problem (5) can be solved in a finite number of steps since it is convex. If problem (5) has a solution, then the objective value $F(x, y(x))$ is the fitness of individual l and l is called a feasible individual; otherwise, l is called an unfeasible individual, and we set its fitness value be large enough.

3.3. Crossover and mutation operators

Let $l = (0, \ldots, 1, 0, \ldots, 1, \ldots, 1, \ldots, 0)$ be a selected parent for crossover. We choose randomly a component 1 of l, and change it to 0. In order to keep the number of basic variables unchanged, we choose randomly a component 0 of l, and change it to 1. The resulting individual is called the crossover offspring of l.

Let $\bar{l} = (0, \ldots, 1, 0, \ldots, 1, \ldots, 1, \ldots, 0)$ be a selected parent for mutation. We first denote indices of component 1 in $\bar{l}$ by $t_1, t_2, \cdots, t_q$, and set $1 \leq t_1 \leq t_2 \cdots \leq t_q \leq (m + q)$. Further, we take integers $\bar{l}_{o1} \in [1, t_2)$, $\bar{l}_{oj} \in [t_j, t_{j+1})$, where $j = 2, \cdots, q - 1$, and $\bar{l}_{oq} \in [t_q, t_{m+q}]$. At last, the mutation offspring $\bar{l}_o$ of $\bar{l}$ can be got by taking the $\bar{l}_{oi}$-th component as 1, $i = 1, 2, \cdots, q$, and other components as 0.

Evidently, the crossover operator is designed to search extreme points adjacent to that represented by l, while the mutation operator is to search some points far away from the point represented by $\bar{l}$.

3.4. A technique for B^{-1}

It should be noted that for large scale linear programming, if the traditional method is used to get B^{-1}, it will cause a large amount of computation. So it is necessary to develop an efficient technique to overcome the disadvantage. It is well-known that the crossover probability, generally speaking, is much larger than the mutation probability, therefor, the crossover can generate most offspring while the mutation operator generates a small number of offspring. As a result, we are devoted to finding a method to get B^{-1} for the crossover offspring. At first, in the initial population, for each individual, one has to use traditional methods to get B^{-1}. But for the crossover offspring, we can get the B^{-1} from the inverse matrix associated with the crossover parent.

Let l be a parent for crossover, l' is its crossover offspring, and

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m+q} \\ a_{21} & a_{22} & \dots & a_{2m+q} \\ \dots & \dots & \dots & \dots \\ a_{q1} & a_{q2} & \dots & a_{qm+q} \end{pmatrix}$$

with any loss of generality, let the base corresponding to l be B and

$$B = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1q} \\ a_{21} & a_{22} & \dots & a_{2q} \\ \dots & \dots & \dots & \dots \\ a_{q1} & a_{q2} & \dots & a_{qq} \end{pmatrix}, \quad B^{-1} = \begin{pmatrix} a'_{11} & a'_{12} & \dots & a'_{1q} \\ a'_{21} & a'_{22} & \dots & a'_{2q} \\ \dots & \dots & \dots & \dots \\ a'_{q1} & a'_{q2} & \dots & a'_{qq} \end{pmatrix}$$

Let B' be a base corresponding to l', and with any loss of generality, set

$$B' = \begin{pmatrix} a_{11} & \dots & a_{1,i-1} & a_{1,i+1} & \dots & a_{1q} & a_{1j} \\ a_{21} & \dots & a_{2,i-1} & a_{2,i+1} & \dots & a_{2q} & a_{2j} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_{q1} & \dots & a_{q,i-1} & a_{q,i+1} & \dots & a_{qq} & a_{qj} \end{pmatrix}$$

where $1 \le i \le q$ and $q < j \le m + q$. Then B'^{-1} can be got by using following algorithm.

Algorithm 1

Step 1. Compute $A'_j = B^{-1}a_{.j}$, where $a_{.j}$ is the j-th column of A. Set $A'_j = (\lambda_{1j}, \dots, \lambda_{qj})^T$. If $\lambda_{ij} = 0$, then B' is singular; otherwise, return to Step 2;
Step 2. Let $\Lambda = (e_1, \dots, e_{i-1}, \bar{\lambda}, e_{i+1}, \dots, e_q)$, where $e_j \in R^q$ is the j-th column of unit matrix, $j = 1, \dots, q$. $\bar{\lambda} = (-\frac{\lambda_{1j}}{\lambda_{ij}}, \dots, -\frac{\lambda_{i-1,j}}{\lambda_{ij}}, \frac{1}{\lambda_{ij}}, -\frac{\lambda_{i+1,j}}{\lambda_{ij}} \dots, -\frac{\lambda_{qj}}{\lambda_{ij}})^T$;
Step 3. Compute $\bar{B}' = \Lambda B^{-1}$;
Step 4. $B'^{-1} = (b'_{1.}, \dots, b'_{i-1,.}, b'_{i+1,.}, \dots, b'_{q.}, b'_{i.})^T$, where $b'_{i.}$ is the i-th row of $\bar{B}'$.

3.5. Genetic algorithm based on the optimality conditions(GA/OC)

Step 1. (*Initialization*) Randomly generate N initial points $\{l^i \in R^{m+q} | i = 1, 2, \cdots, N\}$, in which only q components are 1 for each l^i, while other components are 0. All of the points l^i form the initial population $pop(0)$ with population

size N. Let $k = 0$.
Step 2. (*Fitness*) Evaluate the fitness value $F(x, y)$ of each point in $pop(k)$.
Step 3. (*Crossover*) Let the crossover probability be p_c. For each $l \in pop(k)$, we first take a $r \in [0, 1]$ at random, if $r \leq p_c$, then execute the crossover on l to get its offspring l_o. Let $O1$ stand for the set of all these offspring.
Step 4. (*Mutation*) Let the mutation probability be p_m. For each $l' \in pop(k)$, we first take a $r \in [0, 1]$ at random, if $r \leq p_m$, then execute the mutation on l' to get its offspring l'_o. Let $O2$ stand for the set of all these offspring.
Step 5. (*Selection*) Let $O = O1 \cup O2$. Evaluate the fitness values of all points in O. Select the best N_1 points from the set $pop(k) \cup O$ and randomly select $N - N_1$ points from the remaining points of the set. These selected points form the next population $pop(k + 1)$.
*Step*6. If the termination condition is satisfied, then stop; otherwise, let $k = k+1$, go to *Step* 3

Remark 1. Compared with the algorithms which begin with the upper-level problem or use K-K-T conditions to replace the follower's problem, the proposed GA/OC shows at least three advantages:

1). GA/OC has much smaller search space than these algorithms do, since the number of the extreme points is not larger than C^q_{q+m}.

2). When the fitness value is computed, the traditional optimization technique is incorporated to GA/OC, which improve the local search ability of the algorithm.

3). Since GA/OC begins with the follower's problem and can obtain its optimal solution easily, it is more efficient than the existing algorithms for BLPPs with large-scale follower's problems.

4. Simulation results

In order to analyze the performance of GA/OC, a computational experiment is performed. With this experiment we aim to show the efficiency of our method in terms of the quality of the solution, including the first objective function value and the computational time involved. The computational study is divided into two parts. The first part is devoted to studying the effect of different parameters of the algorithm on the quality of the solution, and selecting the most efficient configuration for the proposed algorithm. Once this configuration is chosen, the second part of the study goes on to measure how good the obtained solutions are, and compare the algorithm with other existing ones.

4.1. Test problems and configuration of parameters

In this subsection, the values of crossover probability and mutation probability as well as population size are given at two different levels. In order to find out the most efficient configuration of parameters for the proposed algorithm, we have to run the algorithm with every group of parameter values on different test problems. For this purpose, we must first construct some BLPPs satisfying the

TABLE 1. Test problem dimensions for linear case

$G1$			$G2$			$G3$		
n	m	q	n	m	q	n	m	q
28	24	12	42	36	18	70	60	30
28	32	20	42	48	30	70	80	50
28	44	32	42	66	48	70	110	80
20	32	12	30	48	18	50	80	30
20	40	20	30	60	30	50	100	50
20	52	32	30	78	48	50	130	80
8	44	12	12	66	18	20	110	30
8	52	20	12	78	30	20	130	50
8	64	32	12	96	48	20	160	80

proposed conditions. The constructed problems are divided into two types, one is linear BLPP and the other is nonlinear convex BLPP with linear follower's problem(for the purpose of simplicity, the follower's constraints are taken as equations). At first, the linear BLPPs are were randomly generated using the MATLAB environment according to the methods given in [3], in which the test problems are divided into 3 groups denoted by G1, G2 and G3, as shown in Table 1. There are 9 test problems for each group, and total 27 test problems are constructed according to Table 1.

For nonlinear cases, all functions are generated by using the same techniques as above except for the leader's objective function. We generate a leader's objective formulated as

$$F(x,y) = \frac{1}{2}x^T Hx + Ky$$

where $H = (h_{ij})_{n\times n}$ is a positive definite matrix, and each element in H is taken randomly in $(0, 10)$; K is generated from a uniform distribution on $(-10, 10)$. The dimensions of these test problems are given at different levels, as shown in Table 2.

In Table 2, the test problems are also divided into three groups with respective 50, 100 and 150 variables, denoted by G1, G2 and G3. For each group, we build test problems as follows, the numbers of follower's variables are 40%, 60% and 80% of the total $(n + m)$ of all variables, respectively; while the numbers of linear constraints are 40%, 60% and 80% of the total m of follower's variables, respectively.

Note that we constructed two types of test problems, linear BLPPs and nonlinear convex BLPPs with linear follower's problems. In order to find out the more efficient configuration of parameters for GA/OC, for each parameter, two levels are taken into account, that is, the crossover probability $p_c = 0.8$ and 0.5; the mutation probability $p_m = 0.5$ and 0.2; and the population size N = 100 and 50. For each level (parameter value) combination we execute GA/OC on two types of the test problems generated according to G1, linear and nonlinear

TABLE 2. Test problem dimensions for nonlinear case

$G1: n+m=50$			$G2: n+m=100$			$G3: n+m=150$		
n	m	q	n	m	q	n	m	q
30	20	8	60	40	16	90	60	24
30	20	12	60	40	24	90	60	36
30	20	16	60	40	32	90	60	48
20	30	12	40	60	24	60	90	36
20	30	18	40	60	36	60	90	54
20	30	24	40	60	48	60	90	72
10	40	16	20	80	32	30	120	48
10	40	24	20	80	48	30	120	72
10	40	32	20	80	64	30	120	96

TABLE 3. Results at different levels of parameters

No.	p_c	p_m	population size	ST		
				linear case	nonlinear case	total
1	0.5	0.2	50	45	45	90
2	0.5	0.2	100	40	42	82
3	0.5	0.5	50	39	38	77
4	0.5	0.5	100	44	43	87
5	0.8	0.2	50	38	45	83
6	0.8	0.2	100	37	37	74
7	0.8	0.5	50	36	30	66
8	0.8	0.5	100	36	41	77

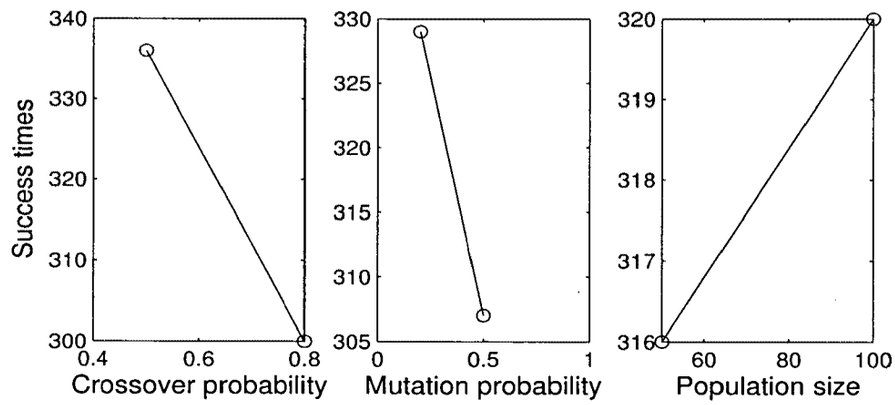

FIGURE 1. Success times at different levels of parameters in total 360 runs

BLPPs, and for each problem GA/OC runs 5 times independently. When total 10000 individuals are searched, the algorithm stops. After doing so, for each combination of levels total 2(types)× 9 (problems with different dimensions)×

ANOVA Table

Source	SS	df	MS	F	Prob>F
Columns	0.01185	1	0.01185	1.28	0.2638
Error	0.48295	52	0.00929		
Total	0.4948	53			

FIGURE 2. Analysis of variance for SR provided by GA/OC and GABB

5 (running times) problems are solved. In order to measure the performance of GA/OC, for each test problem and level combination we define as a success, the fitness value being equal to the best value obtained in all dependent runs. e.g. if there are 4 minimal fitness values in 5 runs, the success times(ST) is 4, and the success rate(SR) is 0.8. Table 3 shows the success times for each type and level combination.

In order to illustrate graphically the influence of factors, based on the data in Table 3, we calculate the totals of ST for each level of parameters, e.g. for the crossover probability $p_c = 0.5$, the total of ST is 336 $(90 + 82 + 77 + 87)$. Fig 1 displays the plot of ST at different levels for each parameter, which means the three parameters are significant, especially, the crossover and mutation operators, since they show the evident difference at different levels. But one can't select the parameter values only according to Fig. 1, since it does not include the influences of interaction among the parameters. In fact, based on the statistics, it is reasonable to select a level combination with the maximal ST according to Table 3. As a result, we select $p_c = 0.5, p_m = 0.2$, and $N = 50$.

4.2. Results and discussion

For all constructed test problems, linear and nonlinear ones, We execute GA/OC in 10 independent runs on each problem on a computer(Intel Pentium IV-2.66GHz), and record ST as well as SR. It means GA/OC is executed 2 (types)$\times$ 3(groups)$\times$9 (problem number in each group)$\times$10 (run times) times. For the linear case, [3] gave an efficient algorithm called GABB, and set the stopping condition at 200,300,and 300 iteration for G1, G2, and G3, respectively. Based on the parameter values $(p_c = 0.5, p_m = 0.25, N = 100)$ given in [3], one can know that GABB generated at least the mean of 20000 sample points (=100(population size) $\times$ 0.5 (crossover probability)$\times$2 (crossover offspring) $\times$ 200 (generations)) for G1, and the mean of 30000 sample points (=100(population size) $\times$ 0.5(crossover probability)$\times$ 2 (crossover offspring) $\times$ 300 (generations)) for G2 and G3, respectively. We stop GA/OC when 10000 individuals are generated for G1, while for G2 and G3, the value is 30000. All data are shown in Table 4 and 5.

From Table 4, GA/OC can give a higher rate of success for G1 ,G2, and G3, as did GABB. It means GA/OC is stable and reliable in solving large-scale linear BLPPs. In order to compare the performance of the two algorithms, we make an

TABLE 4. Comparison of the results for linear BLPPs with different dimensions

No.	Group	Dim. & Cons.	CPU(s)	SR(ST)	
				GA/OC	GABB
1	$G1$	$28-24-12$	25	1.00(10)	0.97
2	$G1$	$28-32-20$	35	1.00(10)	0.97
3	$G1$	$28-44-32$	54	1.00(10)	0.93
4	$G1$	$20-32-12$	26	1.00(10)	0.97
5	$G1$	$20-40-20$	38	1.00(10)	1.00
6	$G1$	$20-52-32$	56	1.00(10)	0.93
7	$G1$	$8-44-12$	27	1.00(10)	1.00
8	$G1$	$8-52-20$	37	1.00(10)	1.00
9	$G1$	$8-64-32$	53	1.00(10)	0.90
10	$G2$	$42-36-18$	107	1.00(10)	0.97
11	$G2$	$42-48-30$	150	1.00(10)	0.93
12	$G2$	$42-66-48$	229	1.00(10)	0.87
13	$G2$	$30-48-18$	100	1.00(10)	0.90
14	$G2$	$30-60-30$	150	1.00(10)	0.97
15	$G2$	$30-78-48$	229	1.00(10)	0.83
16	$G2$	$12-66-18$	99	1.00(10)	0.93
17	$G2$	$12-78-30$	153	1.00(10)	0.93
18	$G2$	$12-96-48$	224	1.00(10)	0.90
19	$G3$	$70-60-30$	153	0.70(7)	0.90
20	$G3$	$70-80-50$	236	0.90(9)	1.00
21	$G3$	$70-110-80$	372	0.70(7)	0.90
22	$G3$	$50-80-30$	147	1.00(10)	0.90
23	$G3$	$50-100-50$	224	0.60(6)	0.70
24	$G3$	$50-130-80$	367	0.70(7)	0.90
25	$G3$	$20-110-30$	117	1.00(10)	0.90
26	$G3$	$20-130-50$	222	1.00(10)	0.80
27	$G3$	$20-160-80$	402	1.00(10)	0.90

analysis of variances for the last two columns in Table 4. The results are shown in Fig.2, since $p = 0.2638 > 0.05$, the algorithm, as a factor, has no significant effect on SR, which means GA/OC has the same efficiency as GABB at least on solving linear BLPPs. Fig. 3 displays the boxplot of SR.

Table 5 displays the results for the 27 constructed nonlinear BLPPs. One can see that GA/OC gave more uniform results for all problems, especially for G1 and G2. It should be noted that GABB can't solve the type of BLPPs.

Fig.4 shows the relationship between the CPU time and the scale of problems. We can see that GA/OC needs less time for solving the problems in G1, regardless of linear or nonlinear cases.

5. Conclusion

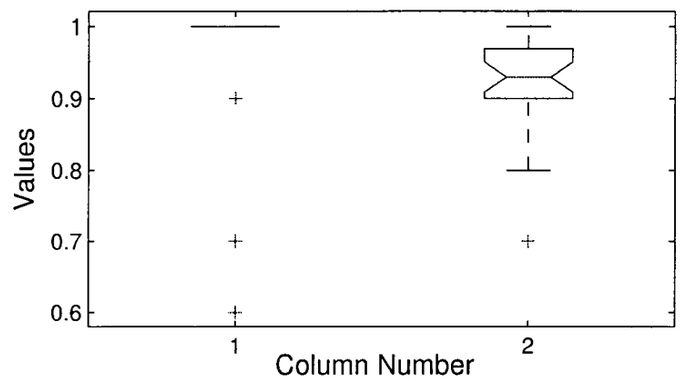

FIGURE 3. Boxplot for SR provided by GA/Oc and GABB

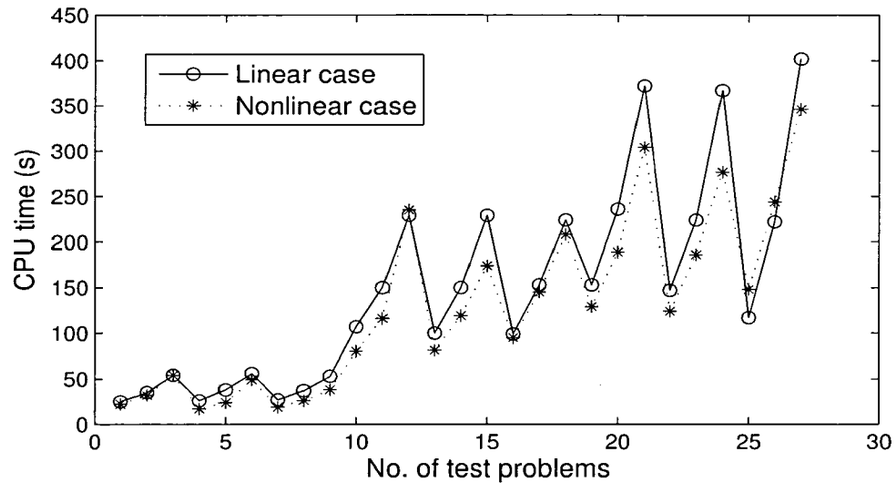

FIGURE 4. CPU time used by GA/OC for linear and nonlinear cases

For the nonlinear bilevel programming problems with convex leader's functions and linear follower's problem, we propose a genetic algorithm based on the optimality conditions of LP(GA/OC). It is evident that for the linear BLPPs, GA/OC has the same efficiency as the compared algorithm, but GA/OC can also solve efficiently the nonlinear BLPPs with convex leader's functions, especially when the follower's problem is a large-scale LP.

6. Acknowledgement

This work was supported in part by the National Natural Science Foundation of China under Grant No. 60873099.

TABLE 5. Results for nonlinear BLPPs with different dimensions

No.	Group	Dim. & Cons.	CPU(s)	SR	ST
1	$G1$	$30-20-8$	22	1.00	10
2	$G1$	$30-20-12$	32	1.00	10
3	$G1$	$30-20-16$	54	1.00	10
4	$G1$	$20-30-12$	17	1.00	10
5	$G1$	$20-30-18$	24	1.00	10
6	$G1$	$20-30-24$	49	1.00	10
7	$G1$	$10-40-16$	19	1.00	10
8	$G1$	$10-40-24$	26	1.00	10
9	$G1$	$10-40-32$	38	1.00	10
10	$G2$	$60-40-16$	80	1.00	10
11	$G2$	$60-40-24$	116	0.90	9
12	$G2$	$60-40-32$	235	1.00	10
13	$G2$	$40-60-24$	81	1.00	10
14	$G2$	$40-60-36$	119	0.70	7
15	$G2$	$40-60-48$	174	0.80	8
16	$G2$	$20-80-32$	95	1.00	10
17	$G2$	$20-80-48$	145	1.00	10
18	$G2$	$20-80-64$	208	1.00	10
19	$G3$	$90-60-24$	129	1.00	10
20	$G3$	$90-60-36$	189	0.80	8
21	$G3$	$90-60-48$	304	0.70	7
22	$G3$	$60-90-36$	124	0.90	9
23	$G3$	$60-90-54$	186	0.70	7
24	$G3$	$60-90-72$	277	0.70	7
25	$G3$	$30-120-48$	148	0.70	7
26	$G3$	$30-120-72$	244	0.80	8
27	$G3$	$30-120-96$	346	0.70	7

REFERENCES

1. E. Aiyoshi and K. Shimizu, *A solution method for the static constrained Stackelberg problem via penalty method*, IEEE Trans. Autom. Control 29 (1984), 1111-1114.
2. J. F. Bard, *Practical Bilevel Optimization*, Kluwer Academic Publishers, The Netherlands, 1998.
3. H. I. Calvete, C. Gale and P. M. Mateo, *A new approach for solving linear bilevel problems using genetic algorithms*, European Journal of Operational Research 188(2008), 14–28.
4. H. I. Calvete and C. Gale, *On the quasiconcave bilevel programming problem*, J. Optimization Theory and Applications 98(1998), 613–622.
5. B. Colson, P Marcotte and G Savard, *Bilevel programming: A survey*, A Quarterly Journal of Operations Research (4OR) 3(2005), 87–107.
6. B. Colson, P. Marcotte and G. Savard, *A trust-region method for nonlinear bilevel programming: algorithm and computational experience*, Computational Optimization and Applications 30(2005), 211-227.

7. Kuen-Ming Lan, Ue-Pyng Wen and Hsu-Shih Shih, *et al, A hybrid neural network approach to bilevel programming problems*, Applied Mathematics Letters 20(2007), 880–884.
8. Hecheng Li and Yuping Wang, *A hybrid genetic algorithm for solving a class of nonlinear bilevel programming problems*, Proceedings of Simulated Evolution and Learning - 6th International Conference, 2006, 408-415.
9. B. D. Liu, *Stackelberg-Nash equilibrium for mutilevel programming with multiple followers using genetic algorithms*, Comput. Math. Appl. 36(1998), 79-89.
10. L D Muu and N V Quy, *A global optimization method for solving convex quadratic bilevel programming problems*, Journal of Global Optimization 26(2003), 199 –219.
11. V. Oduguwa and R. Roy, *Bi-level optimization using genetic algorithm* , Proc. IEEE Int. Conf. Artificial Intelligence Systems, 2002, 123-128.
12. J. Rajesh, K. Gupta and H. S. Kusmakar, *A tabu search based approach for solving a class of bilevel programming problems in chemical engineering*, Journal of Heuristics 9(2003), 307-319.
13. K. Shimizu and E. Aiyoshi, *A new computational method for Stackelberg and minmax problems by use of a penalty method*, IEEE Trans. Autom. Control 26(1981), 460 - 466.
14. H. V. Stackelberg, *The Theory of the Market Economy*, Oxford Univ. Press, Oxford, 1952.
15. H. Tuy, A. Migdalas and N. T. Hoai-Phuong, *A novel approach to bilevel nonlinear programming*, J. Glob. Optim. 38(2007) 527–554.
16. Yuping Wang, Yong-Chang Jiao and Hong Li, *An evolutionary algorithm for solving nonlinear bilevel programming based on a new constraint - handling scheme*, IEEE Trans. on Systems, Man, and Cybernetics-Part C 35(2005), 221-232.
17. Ue-Pyng Wen and Shuh-Tzy Hsu, *Linear bi-Level programming problems–A review*, The Journal of the Operational Research Society 42(1991), 125–133.
18. Xiaobo Zhu, Qian Yu and Xianjia Wang, *A hybrid differential evolution algorithm for solving nonlinear bilevel programming with linear constraints*, Proc. 5th IEEE Int. Conf. on Cognitive Informatics (ICCI'06), 2006, 126–131.
19. Dao Li Zhu, Qing Xua and Zhenghua Lin, *A homotopy methodfor solving bilevel programming problem*, Nonlinear Analysis 57(2004), 917–928.

Hecheng Li received the B.Sc. degree in mathematics from Qinghai Normal University, Xining, China, in 1995, the M.Sc. degrees in mathematics from Northwest Normal University, Lanzhou, China, in 2001, and the Ph.D. degree in applied mathematics from Xidian University, Xi'an, China, in 2009. He is currently with Department of Mathematics and Information Science, Qinhai Normal University, Xining, China. His research interests focus on evolutionalry computation and optimization theory, algorithms and its application.

School of Computer Science and Technology, Xidian University, Xi'an 710071, China; Department of Mathematics and Information Science, Qinghai Normal University, Xining810008, China

e-mail: lihecheng@qhnu.edu.cn

Yuping Wang received the B.Sc. degree in mathematics from Northwest University, Xian, China, in 1983, and the Ph.D. degree in computational mathematics from Xian Jiaotong University, Xian, China, in 1993. Currently, he is a Professor with the School of Computer Science and Technology, Xidian University, Xian, China. His research interests include evolutionary computation, data mining, optimization theory, algorithms, and applications.

School of Computer Science and Technology, Xidian University, Xi'an 710071, China.