Ann Oper Res (2015) 235:543–558
DOI 10.1007/s10479-015-1878-5

# A genetic algorithm using a finite search space for solving nonlinear/linear fractional bilevel programming problems

**Hecheng Li**[1]

Published online: 3 May 2015


**Abstract** The bilevel programming problem is strongly NP-hard and non-convex, which implies that the problem is very challenging for most canonical optimization approaches using single-point search techniques to find global optima. In the present paper, a class of nonlinear bilevel programming problems are considered where the follower is a linear fractional program. Based on a novel coding scheme, a genetic algorithm with global convergence was developed. First, potential bases of the follower's problem were taken as individuals, and a genetic algorithm was used to explore these bases. In addition, in order to evaluate each individual, a fitness function was presented by making use of the optimality conditions of linear fractional programs. Also, the fitness evaluation, as a sub-procedure of optimization, can partly improve the leader's objective. Finally, some computational examples were solved and the results show that the proposed algorithm is efficient and robust.

**Keywords** Bilevel programming · Genetic algorithm · Optimal solutions · Bases

## 1 Introduction

A bilevel programming problem (BLPP) is a hierarchical optimization problem consisting of two levels, in which the first level is dominant over the second level. Consequently, the bilevel programming problem can be divided into two optimization problems located at different levels, the leader's and the follower's problems. Unlike other mathematical programs, the constraints of the bilevel programming problem always involve the optimality to the follower's problem. In other words, any feasible solution must satisfy the requirement that the follower's variable values are optimal to the follower's problem when the leader's variables are fixed. The problem can be formulated as follows

✉ Hecheng Li
hclwxy@aliyun.com

[1] Department of Mathematics, Qinghai Normal University, Xining 810008, China

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

where $x \in R^n$, $y \in R^m$. In this problem,

$$\begin{cases} \min\limits_{x \in X} F(x, y) \\ s.t.\ G(x, y) \leq 0. \end{cases} \tag{2}$$

and

$$\begin{cases} \min\limits_{y \in Y} f(x, y) \\ s.t.\ g(x, y) \leq 0 \end{cases} \tag{3}$$

are the leader's and the follower's problems, respectively. The variables of problem (1) are divided into the leader's variables $x = (x_1, \ldots, x_n)^T$ and the follower's variables $y = (y_1, \ldots, y_m)^T$. Similarly, $F$ and $f$ are known as the leader's and the follower's objective functions respectively, whereas 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 the 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.

The decision-making process of (1) can be stated as follows: The leader first selects a strategy $x$ to optimize his/her objective, then the follower observes the leader's selection and finds a strategy $y$ to optimize his/her own objective. When such pair $(x, y)$ satisfies the leader's constraints, it is called feasible in a bilevel decision-making procedure. The purpose of solving (1) is to optimize $F(x, y)$ on the set of all feasible points.

In the optimization procedure, if the follower has more than one optimal solution for some $x$, decision-makers have to select one in the optimal solution set to match $x$. Two extreme cases are optimistic and pessimistic models (Dempe 2002, 2003). In the paper, we assume that the follower has a unique solution for any $x$, which can make problem (1) stable (Bard 1998).

Over the past 20 years or so, the field of bilevel optimization has received a lot of attention in developing efficient algorithms and applying bilevel models to deal with hard real-world problems.

When all functions involved in (1) are linear, it is called a linear bilevel programming problem (Bard 1998; Calvete et al. 2008; Glackin et al. 2009). Some classical optimization techniques, such as "$k$-th best" algorithms, branch-and bound approaches, base-enumerating methods and penalty methods, etc, have been proposed for this kind of problems (Dempe 1987; Bard 1998; Colson et al. 2007). For nonlinear BLPPs, most of researches so far focus on convex and differential BLPPs in which all functions involved are convex and twice continuously differential (Colson et al. 2007, 2005; Andreani et al. 2009; Dempe 2002; Mersha and Dempe 2011; Wang et al. 2010), especially on convex quadratic BLPPs (Etoa 2010, 2011; Muu and Quy 2003). In addition, other types of BLPPs have been dealt with, e.g., bilevel multiobjective programming problems in which $F$ and/or $f$ are vector-valued functions (Deb and Sinha 2009), and bilevel (mixed-) integer programs where some variables are restricted into the integer set (Gümüs and Floudas 2005; Li and Wang 2008b; Shim et al. 2013), etc.

Applications have been stimulating factors for the development of BLPPs. Some interesting results have been presented for dealing with a variety of intractable problems in real world, such as the investigation of network of oligopolies (Dempe 2003; Abdou-Kandil and Bertrand 1987), the traffic planning (Migdalas 1995; Calvete et al. 2011), the road pricing problem (Dempe and Zemkoho 2012), the tax credits problem for biofuel production (Bard et al. 2000), the network design problem (Ben-Ayed et al. 1988), and the terrorist threat problem (Scaparra and Church 2008; Arroyo and Galiana 2009). Other applications of BLPPs can be found in Dempe (2003), Bard (1998), Colson et al. (2007).

In spite of the fact that there are lots of theoretic results and efficient methods for BLPPs, it does not mean the problem can be solved easily. From a computational point of view, the computational complexity of the problem can be analyzed by the following three items (Dempe 2003; Bard 1998; Colson et al. 2007).

- BLPP is strongly NP-hard;
- Feasible region is non-convex, making the problem non-convex;
- Solution functions of the follower's problem may be non-differential, it implies that BLPP is non-differential.

Item 1 shows this class of problems are intrinsically hard to solve. Items 2–3 mean the algorithmic approaches based on single-point search can't find out the globally optimal solutions very well. The fact leads researchers to adopt some intelligent algorithms using population-search which don't put any requirements for the differentiability and convexity of functions, e.g. evolutionary algorithms(EAs)/genetic algorithms(GAs) (Calvete et al. 2008; Wang et al. 2005; Li and Wang 2008a, 2011; Calvete et al. 2009; Wang et al. 2011), and artificial neural network(ANN) (Lan et al. 2007). etc. These algorithms can be divided into three classes. The first always begins with leader's variables, and for each selected value of $x$, to solve the follower's problem for $y$ (Li and Wang 2008a; Wang et al. 2011). The second class of approaches uses some techniques to transform BLPP into a single-level program, such as penalty functions or K-K-T conditions (Wang et al. 2005). The third is to design the algorithm using problem-specific optimality results (Dempe 1987; Calvete and Gale 1998; Calvete et al. 2009, 2008; Li and Wang 2011). When a BLPP is large-scale, the first and the second classes of algorithms are time-consuming since there are too large search spaces to explore. Using the optimality results of linear/linear fractional BLPPs, Calvete proposed base-based GAs for these two classes of problems (Calvete et al. 2008, 2009), and the experiment results show that it is efficient in solving large-scale problems. It implies the third seems to be promising for solving large-scale problems. Unfortunately, the methods can't deal with other BLPPs.

In the present paper, we deal with a special class of BLPPs in which the follower is a fractional program, whereas the leader's problem is simply solvable. Obviously, the model is more general than the linear and fractional BLPPs discussed by Calvete et al. (2008, 2009). We present a GA-based global optimization algorithm for the bilevel programming problem. First, the proposed algorithm begins with the bases of the follower's problem, and these bases are taken as individuals of GA. Given any individual (base), we consider the set of values of $x$ on which the base is always feasible and optimal. For any $x$ in the set, the follower's solution $y(x)$ can be represented as a linear function of $x$. Then, an optimization method is adopted to find out the best $\bar{x}$ in the set such that $F$ is minimal, and the objective value $F(\bar{x}, y(\bar{x}))$ is taken as the fitness value of the individual. After all bases are checked, the bilevel programming problem is solved. Since the number of bases will increase fast as the scale of the problem become larger, the algorithms using completely enumerating schemes are far from being efficient. In our algorithm, an efficient GA is designed and used to explore these bases.

The proposed algorithm is different from the existing algorithms mainly in two ways: (1) our algorithm begins with the follower's problem and the search space is the set of feasible bases of the follower's program. It means that the search space of the algorithm is finite and smaller than those of other algorithms (Calvete et al. 2008, 2009; Deb and Sinha 2009; Wang et al. 2005, 2011); (2) when the fitness is evaluated, a sub-procedure of optimization is implicitly executed by which the leader's objective value can be improved.

This paper was organized as follows. Some notations and discussed problem were stated in Sects. 2 and 3 gave a profile of algorithmic approach. Genetic operators were designed in Sect. 4, and then based on these operators, Sect. 5 presented our algorithm. In Sect. 6, the convergence was analyzed, and some computational examples were given and solved in Sect. 7. We finally concluded our paper in Sect. 8.

## 2 Discussed problem and basic notations

In this paper a specific BLPP is considered, that is, the follower is a linear fractional program and the leader's problem is solvable. Let us denote the problems by

$$\begin{cases} \min\limits_{x \in X} F(x, y) \\ s.t.\ G(x, y) \leq 0 \\ \min\limits_{y} f(x, y) = \dfrac{c_1 x + d_1 y + e_1}{c_2 x + d_2 y + e_2} \\ s.t.\ Ax + By \leq b, \quad y \geq 0 \end{cases} \tag{4}$$

where $A$ is a $q \times n$-matrix, $B$ is a $q \times m$-matrix, and $b \in R^q$. $X$ is a box set as follows:

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

where $l_i, u_i$ are real constants.

Now we introduce some related definitions and notations as follows Bard (1998).

1) Constraint region: $S = \{(x, y) | x \in X, G(x, y) \leq 0, Ax + By \leq b, \quad y \geq 0\}$.
2) Feasible region of follower's problem for $x$ fixed: $S(x) = \{y | Ax + By \leq b, \quad y \geq 0\}$.
3) Projection of $S$ onto the leader's decision space: $S(X) = \{x \in X | \exists y, (x, y) \in S\}$.
4) Follower's rational reaction set for each $x \in S(X)$: $M(x) = \{y | y \in argmin\{f(x, y), v \in S(x)\}\}$.
5) Inducible region: $IR = \{(x, y) \in S | y \in M(x)\}$.

In terms of aforementioned definitions, problem (4) can also be written as:

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

**Definition 1** (Dempe 1987) (*Feasible solution*) $(x, y)$ is said to be a feasible solution to (4) if and only if $(x, y) \in IR$.

**Definition 2** (Dempe 1987) (*Optimal solution*) $(x^*, y^*)$ is said to be an optimal solution to (4) if $(x^*, y^*) \in IR$ and satisfies

$$F(x^*, y^*) \leq F(x, y), \quad \forall (x, y) \in IR.$$

In the remainder, we always assume that

A1. For all decisions taken by the leader, each follower has some room to react, that is, $S(x) \neq \phi$.
A2. The rank of matrix $B$ is q.
A3. $c_2x + d_2y + e_2 \neq 0$ for $\forall x \in X$.

In these assumptions, A1 is a common assumption for bilevel programming problems, which makes the bilevel programming problem well posed. Assumptions A2 and A3 are often presented in algorithmic approaches to linear fractional programs (Calvete et al. 2009; Swarup 1965), which can simplify the description of algorithms.

Also, in order to solve problem (4) easily, we further assume the leader's problem can be easily solved, that is to say, there exists at least one deterministic approach by which one can obtain the optima of the leader's problem. There are lots of problems of this type, such as linear programs, convex quadratic programs, and linear fractional programs, etc. In fact, the assumption is necessary to almost all bilevel program solvers, otherwise, problem (1) can not be easily solved.

## 3 Optimality conditions and profile of the proposed approach

First, we take advantage of the characteristics of the follower's problem to present some optimality results. Then, based on these results, we provide an algorithmic profile for solving BLPP (4).

### 3.1 Transformation of BLPP

Since linear inequalities can be converted to equalities by adding some slack variables at left-hand side. Without any loss of generality, (4) can be rewritten as follows

$$\begin{cases} \min\limits_{x \in X} F(x, y) \\ s.t.\ G(x, y) \leq 0 \\ \min\limits_{y} \dfrac{c_1x + d_1y + e_1}{c_2x + d_2y + e_2} \\ s.t.\ Ax + By = b, \quad y \geq 0 \end{cases} \tag{5}$$

in which the follower is a linear fractional programming problem with parameter vector $x$ as follows

$$\begin{cases} \min\limits_{y} \dfrac{c_1x + d_1y + e_1}{c_2x + d_2y + e_2} \\ s.t.\ By = b - Ax, \quad y \geq 0. \end{cases} \tag{6}$$

Also, $B$ is still a $q \times m$-matrix.

### 3.2 Optimality results

In the subsection, at first, the optimality results of the linear fractional problems are discussed. Then, these optimality results are used to deal with BLPP (5).

**Theorem 1** (Swarup 1965) *For each $x \in S(X)$, the optimal value of (6) can occur at an extreme point of $S(x)$.*

Theorem 1 implies that one can find the optima of (6) for each $x$ by enumerating all bases associated with extreme points. Next, we present an optimality criterion by which one can judge whether a basic feasible solution is optimal.

Let $B = (\bar{B}, N)$, and without loss of generality, $\bar{B}$, as a base, is composed of the first $q$ columns of $B$. Then $y_{\bar{B}} = \bar{B}^{-1}(b - Ax) - \bar{B}^{-1}Ny_N$. Further,

$$\begin{aligned} & c_i x + d_i y + e_i \\ &= d_{i\bar{B}} y_{\bar{B}} + d_{iN} y_N + c_i x + e_i \\ &= d_{i\bar{B}} \bar{B}^{-1}(b - Ax) + c_i x + e_i + (d_{iN} - d_{i\bar{B}} \bar{B}^{-1} N) y_N \end{aligned} \tag{7}$$

Here, $i = 1, 2$. Set $u_0 = d_{1\bar{B}} \bar{B}^{-1}(b - Ax) + c_1 x + e_1$, $v_0 = d_{2\bar{B}} \bar{B}^{-1}(b - Ax) + c_2 x + e_2$, $\mu = d_{1N} - d_{1\bar{B}} \bar{B}^{-1} N$ and $\nu = d_{2N} - d_{2\bar{B}} \bar{B}^{-1} N$. Then the following theorem can be inferred.

**Theorem 2** *For the base $\bar{B}$ and any $x$ fixed , if inequalities $\pi = v_0\mu - u_0\nu \geq 0$ and $\bar{B}^{-1}(b - Ax) \geq 0$ hold, then the base is optimal, and the basic components of the optimal solution are $\bar{B}^{-1}(b - Ax)$, whereas other components are 0.*

*Proof* When the denominator of the objective function is positive in (6), Theorem 2 is true Swarup (1965). For the case that the denominator is negative, both the denominator and the numerator of $f(x, y)$ are multiplied by $-1$. Set $c_i' = -c_i, d_i' = -d_i$, and $e_i' = -e_i$, then $\pi' = v_0'\mu' - u_0'\nu' = -v_0(-\mu) - (-u_0)(-\nu) = v_0\mu - u_0\nu = \pi$. This completes the proof. □

### 3.3 Sub-procedure of optimization

As discussed above, for any base $\bar{B}$ and some $x$'s in $X$, if the following inequalities

$$\bar{B}^{-1}(b - Ax) \geq 0 \tag{8}$$

and

$$\pi \geq 0 \tag{9}$$

are satisfied, then a subregion of $X$ is determined in which for each $x$, $y(x) = (\bar{B}^{-1}(b - Ax), 0)^T$ is the optimal solution to the follower's problem. In order to obtain the optima of $F(x, y)$, we consider the following nonlinear problem

$$\begin{cases} \min\limits_{x \in X} F(x, y(x)) \\ s.t.\, G(x, y(x)) \leq 0, \\ \bar{B}^{-1}(b - Ax) \geq 0, \\ \pi \geq 0. \end{cases} \tag{10}$$

If (10) is further solved, and an optimal solution $x_0$ is obtained, this means that $x_0$ is the best one in all $x$'s to which $\bar{B}$ is feasible and optimal. Such point $(x_0, y(x_0))$, in fact, is a "locally optimal" solution to (5) (quotation mark means it isn't really a locally optimal point in mathematical meaning of neighborhood), and called a base-optimal point. Figure 1 gives an example for illustrating the relationships, where the elliptic region represents $S(X)$ and the total number of bases is 3. First of all, all bases of the follower's problem are denoted by $B_i, i = 1, 2, 3$. Then, the value region of $x$ is divided into three subregions according to these bases, it implies that for any point in Subregion $i$ ($i$ = I, II, III), the optimal base of the follower's problem is $B_i, i = 1, 2, 3$, respectively. Further, $x_i$ is the "best" one in each subregion, which means the points $(x_i, y(x_i)), i = 1, 2, 3$, are base-optimal points. As a result, the optimal solutions to (5) must be one of these basis-optimal points.

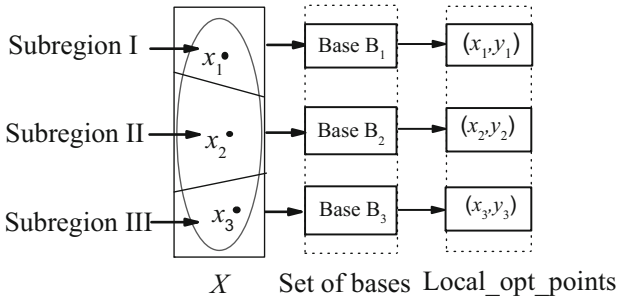



**Fig. 1** Relationships between the bases and the basis-optimal points of BLPP

### 3.4 Profile of algorithmic approach

For any base $\bar{B}$ of the follower's problem, (10) is first solved. If there exists no solution, then the base is removed from the set of all bases. Otherwise, the objective value is used to evaluate the base. When all bases are evaluated, the "best" base can be found, and then (5) is solved. In fact, the base-optimal point corresponding to the "best" base is an optimal solution to (5). In the profile, a genetic algorithm is designed to explore all bases.

## 4 Design of genetic algorithm

In this section, we begin with chromosomes encoding, present the fitness evaluation scheme, and then design crossover and mutation operators.

### 4.1 Chromosome encoding and initial population

Since GA is used to search all bases of the follower's problem, we encode each base of (6) as individual of population. Let $V = \{1, 2, \ldots, m\}$ be the set of all column indices of $B$. $q$ elements are selected from $V$ and denoted by $l = \{i_1, i_2, \ldots, i_q\}$. Furthermore, if these columns are linearly independent, then $l$ is taken as an individual. An initial population with the size of $N_p$ can be generated by lexicographically selecting $N_p$ individuals.

But it is computationally expensive if the determinant method is used for each $l$ to judge whether the selected columns are linearly independent. We present a simplified approach by the following example. Let

$$B = \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \end{pmatrix} \tag{11}$$

Here, $q = 2, m = 4$. For convenience, we denote by $B\{i, j\}$ the matrix consisting of the $i$-th and $j$-th columns of $B$. Without loss of generality, set

$$B\{1, 2\} = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \tag{12}$$

be nonsingular, then we get an individual $l = \{1, 2\}$. Further, we let

$$B\{1, 2\}^{-1} B = \begin{pmatrix} 1 & 0 & \bar{b}_{13} & \bar{b}_{14} \\ 0 & 1 & \bar{b}_{23} & \bar{b}_{24} \end{pmatrix} \tag{13}$$

If $\bar{b}_{23} \neq 0$, then $B\{1, 3\}$ is nonsingular, that is, $l' = \{1, 3\}$ is also an individual, otherwise, $l'$ is ignored. The same procedure can be applied to judge whether $l'' = \{1, 4\}$ is an individual.

If $B\{1, 3\}$ is nonsingular, based on the $B\{1, 2\}^{-1}$, we can utilize the pivoting algorithm to obtain $B\{1, 3\}^{-1}$ as in the simplex method.

### 4.2 Fitness function

For any individual $l = \{l_1, l_2, \ldots, l_q\}$, without loss of generality, the basic matrix associated with $l$ is also denoted by $\bar{B}$. Recall that the leader' problem is solvable, $y(x)$ is linear and the added constraints, $\bar{B}^{-1}(b - Ax) \geq 0$ and $\pi \geq 0$, are also linear, it follows that (10) is solvable with respect to $x$. Hence, some deterministic methods can be selected to solve the problem (10) and the optimal objective value is taken as the fitness of the individual.

### 4.3 Crossover and mutation operators

*Crossover operator* Let $l = \{l_1, l_2, \ldots, l_q\}$ and $l' = \{l'_1, l'_2, \ldots, l'_q\}$ be parents selected for crossover. First, we give a cross-position in $l'$ as in one-point crossover operator. Then the components at left-hand side of the cross position are, one by one, taken as entering elements for $l$ and some components in $l$ are removed as leaving elements, which follows the same procedure as in the simplex method except for the minimum ratio rule, and makes the total of elements in $l$ keep constant. For any entering element which is also in $l$, the replacement is ignored. When all entering procedures are finished, a crossover offspring is generated. When a cross-position is given in $l$, some elements in $l'$ will be replaced and the other offspring can be generated.

As an example, we let $l = \{1, 2\}$ and $l' = \{3, 4\}$ be parents for crossover in the above example, and the cross-position is selected at random as follows.

$$l = \{1 \mid 2\}, \ \ l' = \{3 \mid 4\}$$

It means that 3 should be put into $l$, and 1 or 2 will be removed from $l$. Let us re-check the matrix (13). Any nonzero element is chosen in the 3-th column of the matrix. If $\bar{b}_{i3}$ is chosen, then the $i$-th element in $l$ should be replaced, $i = 1, 2$. after doing so, a crossover offspring is generated.

As discussed in the above subsection, when the replacements are executed one by one, the inverse matrices associated with individuals can be gotten by the pivot algorithm.

In order to generate offspring as well as possible, we always take the best individual as one of parents in each crossover process.

*Mutation operator* Let $\bar{l} = \{l_1, l_2, \ldots, l_q\}$ be a parent individual selected for mutation, and $\tilde{B} = B\{l_1, l_2, \ldots, l_q\}$. First, generate randomly a integer $z(1 \leq z \leq q)$ and select $z$ indices from the set $V \backslash \bar{l}$, and then put these indices, one by one, into $\bar{l}$ as done in the crossover operation. As a consequence, $z$ indices are replaced, and a mutation offspring is obtained.

## 5 Proposed algorithm

In this section, we present a genetic algorithm using a finite search space(GA-FSS), which is described as follows.

*Step 0* Some parameters are given, population size $N_p$, crossover probability $p_c$, mutation probability $p_m$, $\Re = \phi$ and an integer $K$;

*Step 1* $N_p$ individuals are generated, these individuals form an initial population denoted by $pop(0)$. Let $k = 0$;

*Step 2* The fitness is evaluated for each individual, and the best individual $l_{best}$ in $pop(k)$ is recorded with its fitness value $F_{best}$. These individuals with fitness values are put into $\Re$;

*Step 3* Crossover is applied to each individual in $pop(k)$ according to crossover probability $p_c$, and crossover offspring set is denoted by $O_c$;

*Step 4* Mutation is executed to each each individual in $pop(k)$ according to mutation probability $p_m$, and mutation offspring set is represented as $O_m$;

*Step 5* Evaluate offspring generated by crossover and mutation operators. If some offspring belong to $\Re$, the evaluation can be ignored. Select the best $N_p$ individuals from $pop(k) \cup O_c \cup O_m$ as next population $pop(k+1)$, and update $F_{best}$ as well as $l_{best}$. Also, select some offspring for which the fitness evolution is computation-complex, then put these offspring into the set $\Re$ until there are $K$ points.

*Step 6* If the stopping criterion is satisfied, then output $l_{best}$ and $F_{best}$; otherwise, let $k = k + 1$, return to Step 3.

The procedure shows at least two advantages: one is that the search space has at most $C_m^q$ points, which is far smaller than those of most existing algorithms; the other is that there exists a local searching process in the optimization of (10), which, as a sub-procedure of optimization, is helpful for GA-FSS to improve the value of $F$.

Since the design of GA-FSS depends mainly on the follower problem, it can be used to deal with more general BLPPs than the approaches in Calvete et al. (2008, 2009). When the denominator of $f(x, y)$ is 1 and the leader's problem is linear, the problem becomes a linear BLPP, which is the simplest case for GA-FSS.

## 6 Convergence analysis

In order to analyze the convergent of the proposed algorithm, some preliminaries are first given Wang (2011), Bäck (1996):

**Definition 3** (*Monotonic*) If $F(l_{best}(k+1)) \leq F(l_{best}(k))$, then population sequence $\{pop(k), k = 0, 1, 2, \ldots, \}$ is said to be monotonic.

Here, $l_{best}(k)$ stands for the best individual in $pop(k)$, and $F(l)$ is the fitness of individual $l$.

**Lemma 1** (Wang 2011; Bäck 1996) *If a genetic algorithm satisfies: (i) the search space is finite, (ii) population sequence is monotonic, and (iii) for* $\forall l, l' \in \bar{\Omega}$, $\exists p_0 > 0$ *such that* $prob\{l' = mut(l)\} \geq p_0$, *then it converges to a globally optimal solution with probability one.*

Here, $\bar{\Omega}$ stands for the search space, $prob\{A\}$ is the probability of $A$ and $mut(l)$ represents the mutation offspring of $l$.

In the proposed GA-FSS, the search process, in fact, is executed at two levels. The first-level search is to explore the set of potential bases of the follower, whereas the second-level search is to find the best point $(x, y)$ related to each basis. Recall that we always assume that the leader problem is solvable, it implies that one can obtain the optima of (4) once he finds out the 'best' basis in the first-level search. Applying Lemma 1, we have

**Theorem 3** *The proposed GA-FSS converges to global optima with probability one.*

*Proof* Notice that the search space of GA-FSS only includes at most $C_m^q$ points, it leads to that (i) is satisfied. According to the selection operator, the best individuals found so far are always selected for the next generation of population, hence, (ii) is also satisfied. Next, we need to verify that (iii) holds.

Without any loss of generality, let

$$l = (i_1, \ldots, i_s, i_{s+1}, \ldots, i_q), l' = (i_1, \cdots, i_s, j_{s+1}, \ldots, j_q),$$

that is, there are $q - s$ different entries. Now we compute $prob\{l' = mut(l)\}$. First, the probability of selecting $(j_{s+1}, \ldots, j_q)$ from $V \backslash l$ is $\frac{1}{q}\frac{1}{C_{m-q}^{q-s}}$. Here, $\frac{1}{q}$ means the probability of $z$ being taken as $q-s$, i.e., the total $q-s$ indices will be put into $l$, whereas $\frac{1}{C_{m-q}^{q-s}}$ is the probability of $(j_{s+1}, \ldots, j_q)$ being chosen for entering basis. Next, when $j_v (v \in \{s+1, \ldots, q\})$ is put into $l$, we denote by $\hat{p}_v$ the probability of the replaced element belonging to $(i_{s+1}, \ldots, i_q)$, it is obvious that $\hat{p}_v \geq \frac{1}{q}$. It follows that when $j_{s+1}, \ldots, j_q$, one by one, are put into $l$, the probability of $(i_{s+1}, \ldots, i_q)$ being replaced is not less than $\frac{1}{q^{q-s}}$. As a result, we have

$$prob\{l' = mut(l)\} \geq p_m \times \frac{1}{q}\frac{1}{C_{m-q}^{q-s}} \times \frac{1}{q^{q-s}} > 0$$

This completes the proof. □

## 7 Computational experiments

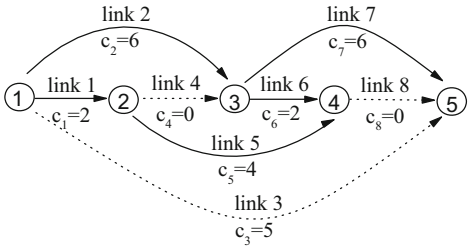
The computational experiments were carried out on two groups of bilevel programming problems with different scales, small and larger-sized problems. First, we executed GA-FSS on 4 small-sized problems which were frequently solved to illustrate the performance of algorithmic methods in literatures. In addition, we randomly generated four moderate-sized problems on which GA-FSS with different parameter configurations was tested. By comparing the computational results, we confirmed the reasonable parameter setting. Finally, GA-FSS with confirmed configuration was applied to solve some larger-sized bilevel programs and computational costs were compared. All computations were executed on a Pentium IV 2.66 processor with 256M RAM running Windows XP.

First, four small scale problems are given as follows.

*Example 1* (Wang et al. 2005)

$$\begin{cases} \min\limits_{x \geq 0} -8x_1 - 4x_2 + 4y_1 - 40y_2 - 4y_3 \\ \min\limits_{y \geq 0} \dfrac{1 + x_1 + x_2 + 2y_1 - y_2 + y_3}{6 + 2x_1 + y_1 + y_2 - 3y_3} \\ s.t. \ -y_1 + y_2 + y_3 + y_4 = 1, \\ 2x_1 + y_1 + 2y_2 - 0.5y_3 + y_5 = 1, \\ 2x_2 + 2y_1 - y_2 - 0.5y_3 + y_6 = 1. \end{cases} \tag{14}$$

**Fig. 2** Network for Toll 1




*Example 2*

$$\begin{cases} \min\limits_{x\geq 0}(-8x_1 - 4x_2 + 4y_1 - 40y_2 - 4y_3 + 29.2)^2 \\ \min\limits_{y\geq 0} \dfrac{1 + x_1 + x_2 + 2y_1 - y_2 + y_3}{6 + 2x_1 + y_1 + y_2 - 3y_3} \\ s.t. \ -y_1 + y_2 + y_3 + y_4 = 1, \\ 2x_1 + y_1 + 2y_2 - 0.5y_3 + y_5 = 1, \\ 2x_2 + 2y_1 - y_2 - 0.5y_3 + y_6 = 1. \end{cases} \tag{15}$$

*Example 3* (Lan et al. 2007)

$$\begin{cases} \max\limits_{x\geq 0} -2x + 11y \\ \max\limits_{y\geq 0} -x - 3y \\ s.t.\ x - 2y \leq 4, 2x - y \leq 24, 3x + 4y \leq 96, \\ x + 7y \leq 126, -4x + 5y \leq 65, x + 4y \geq 8. \end{cases} \tag{16}$$

*Example 4* (Toll 1 in Colson et al. (2005), a toll-setting problem) The network and the costs are shown in Figure 2. Nodes 1 and 5 constitute its unique origin-destination pair. In this problem, $n = 3, m = 8, p = 3$, and $q = 13$. For the more detailed description of the problem, refer to Colson et al. (2005).

For Example 1, we enumerated all 20 potential bases, in which only 4 bases are available. The optimal objective value is $-29.2$. Example 2 is as same as Example 1 except that $F$ is nonlinear. Obviously, the problem has the same optimal solution as Example 1, and the optimal value is 0. Example 3 is a linear programming problem, the optimal objective value reported in the literature is 85.0855. As a real-world example, the reported maximal profit of Example 4 is 7.

For these small-scale problems, the parameters were given as follows: $p_c = 0.8$, $p_m = 0.1$, $N_p = 5$ and $K = N_p * 3$. The maximum number of generations(MaxG) was taken as 10. For each example, 10 independent runs were executed, the computational results were presented in Table 1. In this Table, $\bar{F}$ and $std$ stand for the mean of objective values and standard deviation in 10 runs, respectively. Also, in order to measure the convergent speed of GA-FSS, we considered the number of generations(G) and CPU time(T), and recorded the means of G and T. These two means are represented by $\bar{G}$ and $\bar{T}$, respectively. In column $\bar{T}$, the numbers in square brackets are CPU time needed by the compared algorithms(Despite fact that CPU time was obtained on different hardware equipments, we listed them in the table as a reference). "−" means the item does not exist.

**Table 1** Computational results by GA-FSS and comparison

| No. | GA-FSS | | | | Compared algorithms | |
|---|---|---|---|---|---|---|
| | $\bar{F} \pm std$ | $\bar{G}$ | $\bar{T}(s)$ | Solutions | $F^*$ | Methods |
| 1 | $-29.2 \pm 0$ | 2.3 | 0.81[107] | (0, 0.9, 0, 0.6, 0.4, 0, 0, 0) | $-29.2$ | EA |
| 2 | $0 \pm 0$ | 2.1 | 0.95 | (0, 0.9, 0, 0.6, 0.4, 0, 0, 0) | 0 | – |
| 3 | $85.0909 \pm 0$ | 1.7 | 0.40 | (17.4545, 10.9091) | 85.0855 | ANN |
| 4 | $7 \pm 0$ | 4.7 | 0.57[2.4] | $T_{a \in \mathcal{A}_1} = (7, 9.9, 9.3)$ Optimal path: $1 \rightarrow 5$ | 7 | Trust region |

**Table 2** Mean objective values at different parameter configurations

| No. | $N_p$ | $p_c$ | $p_m$ | $p1$ | $p2$ | $p3$ | $p4$ |
|---|---|---|---|---|---|---|---|
| $c1$ | 50 | 0.5 | 0.05 | $-919.4$ | $-154.2$ | $-351.4$ | $-699.0$ |
| $c2$ | 50 | 0.5 | 0.10 | $-919.4$ | $-255.2$ | $-344.2$ | $-883.9$ |
| $c3$ | 50 | 0.8 | 0.05 | $-919.4$ | $-144.4$ | $-351.4$ | $-755.5$ |
| $c4$ | 50 | 0.8 | 0.10 | $-919.4$ | $-255.2$ | $-351.4$ | $-907.5$ |
| $c5$ | 100 | 0.5 | 0.05 | $-919.4$ | $-144.4$ | $-268.1$ | $-763.2$ |
| $c6$ | 100 | 0.5 | 0.10 | $-919.4$ | $-255.2$ | $-466.1$ | $-909.7$ |
| $c7$ | 100 | 0.8 | 0.05 | $-919.4$ | $-255.2$ | $-382.3$ | $-907.5$ |
| $c8$ | 100 | 0.8 | 0.10 | $-919.4$ | $-154.2$ | $-390.0$ | $-876.0$ |

In addition, in order to obtain a reasonable parameter configuration, we randomly generated 4 moderate-sized bilevel programs with the following type.

$$\begin{cases} \min\limits_{x \geq 0} cx + dy \\ \min \dfrac{c_1 x + d_1 y + e_1}{c_2 x + d_2 y + e_2} \\ s.t.\ Ax + By = b, \quad y \geq 0 \end{cases} \tag{17}$$

All coefficients were generated from uniform distributions. $c, d$ were randomly generated in $[-10, 10]$. The coefficients of fractional objectives were taken as follows: $c_1, d_1$ and $e_1$ were randomly generated in [30,40] and $c_2, d_2, e_2$ were taken randomly in [10, 20]. $A$ and $B$ were generated in $[-10, 10]$ except for the first row generated in $[0, 10]$, which can guarantee the constraint region is bounded. Also, $b$ was obtained by taking the sum of the absolute values of the coefficients of each constraint. Considering that the search space is composed of potential bases of the follower's problem and the leader's problems can be solved determinately, we uniformly took $n = 10$. $m$ was taken as 10, 20, 30 and 40 respectively, and $q$ is 50 % of $m$, then 4 problems with different scales($p1 - p4$) were generated. In GA-FSS, three main parameters are population size $N_p$, crossover probability $p_c$ and mutation probability $p_m$. In order to find a reasonable configuration of these parameters, we consulted the procedure in Calvete et al. (2008) and took two levels for each parameter as follows: $N_p = 50$ or 100, $p_c = 0.5$ or 0.8 and $p_m = 0.05$ or 0.1. Each of parameters has 2 levels, total $2^3$ configurations($c1 - c8$) should be considered, see Table 2. For each configuration, we executed GA-FSS 10 independent runs on each generated problem and when a fixed number of individual evaluations was satisfied, the algorithm was stopped. It follows that total $4 \times 10 \times 8$ runs of GA-FSS need to be executed. The mean objective values of all

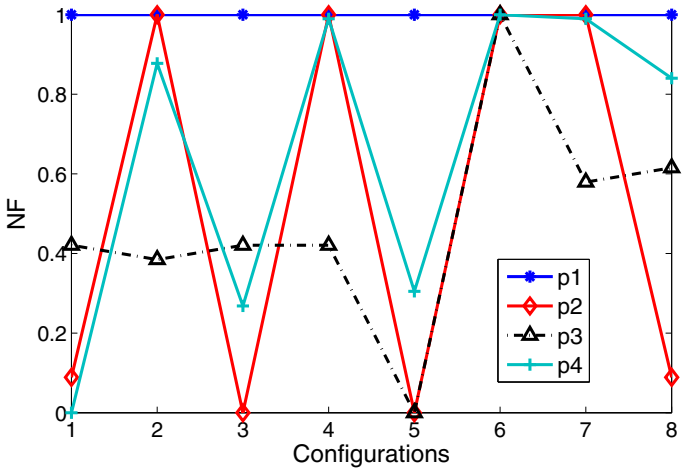



**Fig. 3** Computational results at all parameter configurations

problems ($p1 - p4$) are shown in Table 2. In order to show clearly which is the best one among all configurations, we normalized the data on each of columns $p2 - p4$ by

$$\Lambda_i^* = \frac{\Lambda_{max} - \Lambda_i}{\Lambda_{max} - \Lambda_{min}}$$

Here, $\Lambda = (\Lambda_1, \ldots, \Lambda_8)$ is a column vector, and $\Lambda^*$ is a normalized vector. According to the normalization procedure, the minimum in each column is normalized as 1. For column $p1$, we directly take 1 as normalization result. The process can make all computational results plotted clearly in one figure, see Fig. 3, where $NF$ stands for the normalized values of ojectives. From the figure, one can see that $c6$ is the best one in that all 4 problems achieve 1. As a result, we select the following parameter setting: $N_p = 100$, $p_c = 0.5$ and $p_m = 0.1$.

The third part of the computational experiment is to test the performance of GA-FSS on moderate- and larger-scale problems. In these examples, the maximum number of variables is up to 150, few algorithms were tested on the scale. Using the same procedure as in Calvete et al. (2009), we generated seven types of linear fractional bilevel programming problems, and compared the scales of the search spaces used by GA-FSS and by EPHS(proposed in Calvete et al. (2009), and executed on a PC Pentium 4 at 3.0 GHz having 3.5 GB of RAM), see Table 3, where Num-P represents the number of points in the search space. Obviously, for each problem the search space of GA-FSS is much smaller than that of EPHS. We executed GA-FSS with the selected parameter values 10 independent runs on each problem, and the termination criterion was taken as follows: when GA-FSS can't improve the objective value in successive 50 iterations, the algorithm stops. For each problem there exists no optimal solution as a reference, we took the smallest value in all 10 runs as the best solution to the problem.

In the experiment, for each problem, when the best result appears for the first time, we recorded: (1) the mean of the individual numbers(Num-Ind) evaluated by GA-FSS; and (2) the mean of the CPU time(CPU) invested by the algorithm. All results are shown in Table 4. In order to compare our results with those provided by EPHS, in accordance with EPHS, we calculated the expectancy of the individual numbers required for finding the best solutions. Besides, we listed the numbers(NR) of runs giving the best solutions for each problem.

**Table 3** Scales of problems and comparison of the search spaces used by GA-FSS and EPHS

| No. | Scales | Num-P | |
|---|---|---|---|
| | *n-m-q* | GA-FSS | EPHS |
| 1 | 20-20-16 | 4845 | 6.2e+10 |
| 2 | 12-28-16 | 3.0e+07 | 6.2e+10 |
| 3 | 35-35-28 | 6.7e+06 | 2.7e+19 |
| 4 | 21-49-28 | 3.9e+13 | 2.7e+19 |
| 5 | 50-50-40 | 1.0e+10 | 1.3e+28 |
| 6 | 30-70-40 | 5.5e+19 | 1.3e+28 |
| 7 | 75-75-60 | 2.2e+15 | 4.6e+42 |
| 8 | 45-105-60 | 1.0e+30 | 4.6e+42 |

**Table 4** Mean CPU time(CPU), mean of the individual numbers(Num-Ind) and the number of runs(NR) giving the best solutions

| No. | CPU(s) | | Num-Ind | | NR | |
|---|---|---|---|---|---|---|
| | GA-FSS | EPHS | GA-FSS | EPHS | GA-FSS | EPHS |
| 1 | 4.7 | 2.8 | 1420 | 7722 | 10 | 10 |
| 2 | 5.1 | 3.5 | 2380 | 7457 | 10 | 10 |
| 3 | 17.3 | 58.7 | 2260 | 13,543 | 10 | 9 |
| 4 | 42.1 | 125.9 | 5220 | 15,433 | 10 | 10 |
| 5 | 59.2 | 184.2 | 4560 | 17,285 | 8 | 9 |
| 6 | 172.8 | 747.7 | 7040 | 24,127 | 9 | 10 |
| 7 | 550.4 | – | 7762 | – | 8 | – |
| 8 | 978.3 | – | 10,982 | – | 6 | – |

In Table 4, one can see that the individual number (Num-Ind) required by GA-FSS is far less than that by EPHS, which is mainly due to GA-FSS using a sub-procedure of optimization for the leader objective and having smaller search space than EPHS. For the number of runs(NR) giving the best solutions, based on the analysis of variances, we conclude that the proposed algorithm is almost the same stable and effective as EPHS. In the table "–" means that the values are not provided in the corresponding reference.

We also executed GA-FSS on two kinds of BLPPs with $F(x, y) = cx + dy + e, x \in \{0, 1\}$ and $F(x, y) = (cx + dy + e)^2$ respectively. The computational results show there is no evident difference from the linear fractional case expect for CPU time, which can be easily explained in that the CPU time can be affected by selecting different optimization methods to solve the leader's problems.

Besides, it should be noted that GA-FSS can deal with more general BLPPs than EPHS, for example, when the leader's problem is convex, EPHS can't be used to solve this kind of problems.

## 8 Conclusion

It is very difficult for us to design an efficient algorithm for nonlinear BLPPs with global convergence, especially when the scale of problem is very large. As a consequence, some theoretical results of optimality or the features of problems should be considered in the design

of approaches. In this paper we presented an efficient genetic algorithm by making use of the optimality results of linear fractional programs, and in this algorithmic approach, there are no any restrictions on the leader except that it is solvable. In future work, some BLPPs with other follower will be considered, such as quadratic programming problem, etc.

**Acknowledgments** The research work was supported by the National Natural Science Foundation of China under Grant Nos. 61463045 and 61065009, the Natural Science Foundation of Qinghai Provincial under Grant No. 2013-z-937Q, and the National Social Science Fund of China under Grant No. 13BXW037.

## References

Abdou-Kandil, H., & Bertrand, P. (1987). Government-private sector relations as a stackelberg game: A degenerate case. *Journal of Economic Dynamics and Control*, *11*, 513–517.

Andreani, R., Castro, S. L. C., Chela, J. L., Friedlande, A., & Santos, S. A. (2009). An inexact-restoration method for nonlinear bilevel programming problems. *Computational Optimization and Applications*, *43*, 307–328.

Arroyo, J. M., & Galiana, F. D. (2009). On the solution of the bilevel programming formulation of the terrorist threat problem. *IEEE Transactions on Power Systems*, *20*(2), 789–797.

Bäck, T. (1996). *Evolutionary algorithms in theory and practice*. Oxford: Oxford University Press.

Bard, J. F. (1998). *Practical bilevel optimization: Algorithms and applications*. Dordrecht: Kluwer Academic Publishers.

Bard, J. F., Plummer, J. C., & Sourie, J. C. (2000). A bilevel programming approach to determining tax credits for biofuel production. *European Journal of Operational Research*, *120*, 30–43.

Ben-Ayed, O., Boyce, D., & Blair, C. (1988). A general bilevel linear programming formulation of the network design problem. *Transportation Research*, *22B*, 311–318.

Calvete, H. I., & Gale, C. (1998). On the quasiconcave bilevel programming problem. *Journal of Optimization Theory and Applications*, *98*, 613–622.

Calvete, H. I., Gale, C., & Mateo, P. M. (2008). A new approach for solving linear bilevel problems using genetic algorithms. *European Journal of Operational Research*, *188*, 14–28.

Calvete, H. I., Gale, C., & Mateo, P. M. (2009). A genetic algorithm for solving linear fractional bilevel problems. *Annals of Operations Research*, *166*, 39–56.

Calvete, H. I., Gale, C., & Oliveros, M. (2011). Bilevel model for production-distribution planning solved by using ant colony optimization. *Computers and Operations Research*, *38*, 320–327.

Colson, B., Marcotte, P., & Savard, G. (2007). An overview of bilevel optimization. *Annals of Operations Research*, *153*, 235–256.

Colson, B., Marcotte, P., & Savard, G. (2005). A trust-region method for nonlinear bilevel programming: Algorithm and computational experience. *Computational Optimization and Applications*, *30*, 211–227.

Deb, K., & Sinha, A. (2009). An evolutionary approach for bilevel multi-objective problems. *Communications in Computer and Information Science*, *35*, 17–24.

Dempe, S. (1987). A simple algorithm for the linear bilevel programming problem. *Optimization*, *18*, 373–385.

Dempe, S. (2002). *Foundations of bilevel programming*. Dordrecht: Kluwer Academie Publishers.

Dempe, S. (2003). Annotated bibliography on bilevel programming and mathematical programs with equilibrium constraints. *Optimization*, *52*(3), 333–359.

Dempe, S., & Zemkoho, A. B. (2012). Bilevel road pricing: Theoretical analysis and optimality conditions. *Annals of Operations Research*, *196*, 223–240.

Etoa, J. B. E. (2010). Solving convex quadratic bilevel programming problems using an enumeration sequential quadratic programming. *Journal of Global Optimization*, *47*, 615–637.

Etoa, J. B. E. (2011). Solving quadratic convex bilevel programming problems using a smoothing method. *Applied Mathematics and Computation*, *217*, 6680–6690.

Glackin, J., Ecker, J. G., & Kupferschmid, H. (2009). Solving bilevel linear programs using multiple objective linear programming. *Journal of Optimization Theory and Applications*, *140*, 197–212.

Gümüs, Z. H., & Floudas, C. A. (2005). Global optimization of mixed-integer bilevel programming problems. *Computational Management Science*, *2*, 181–212.

Lan, K. M., Wen, U. P., & Shih, H. S. (2007). A hybrid neural network approach to bilevel programming problems. *Applied Mathematics Letters*, *20*, 880–884.

Li, H., & Wang, Y. (2008a). An interpolation-based genetic algorithm for solving nonlinear bilevel programming problems. *Chinese Journal of Computers*, *31*(6), 910–918.

Li, H., & Wang, Y. (2008b). Exponential distribution-based genetic algorithm for solving mixed-integer bilevel programming problems. *Journal of Systems Engineering and Electronics*, *19*(6), 1159–1164.
Li, H., & Wang, Y. (2011). A real-binary coded genetic algorithm for solving nonlinear bilevel programming with nonconvex objective functions. In *The proceedings of 2011 IEEE congress on evolutionary computation (CEC)* (pp. 2496–2500). New Orleans, USA.
Mersha, A. G., & Dempe, S. (2011). Direct search algorithm for bilevel programming problems. *Computational Optimization and Applications*, *49*, 1–15.
Migdalas, A. (1995). Bilevel programming in traffic planning: Models, methods and challenge. *Journal of Global Optimization*, *7*, 381–405.
Muu, L. D., & Quy, N. V. (2003). A global optimization method for solving convex quadratic bilevel programming problems. *Journal of Global Optimization*, *26*, 199–219.
Shim, Y., Fodstad, M., Gabriel, S. A., & Tomasgard, A. (2013). A branch-and-bound method for discretely-constrained mathematical programs with equilibrium constraints. *Annals of Operations Research*, *210*, 5–31.
Swarup, K. (1965). Linear fractional functional programming. *Operations Research*, *13*(6), 1029–1036.
Scaparra, M. P., & Church, R. L. (2008). A bilevel mixed-integer program for critical infrastructure protection planning. *Computers and Operations Research*, *35*(6), 1905–1923.
Wang, G., Zhu, K., & Wan, Z. (2010). An approximate programming method based on the simplex method for bilevel programming problem. *Computers and Mathematics with Applications*, *59*, 3355–3360.
Wang, Y., Jiao, Y. C., & Li, H. (2005). An evolutionary algorithm for solving nonlinear bilevel programming based on a new constraint-handling scheme. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, *35*(2), 221–232.
Wang, Y., Li, H., & Dang, C. (2011). A new evolutionary algorithm for a class of nonlinear bilevel programming problems and its global convergence. *INFORMS Journal on Computing*, *23*(4), 618–629.
Wang, Y. (2011). *Theory and methodology of evolutionary computation*. Beijing: Science Press.