A solution to bi/tri-level programming problems using particle swarm optimization

Jialin Han[a,b], Guangquan Zhang[a], Yaoguang Hu[b], Jie Lu[a,*]

[a]*Decision Systems and e-Service Intelligence Laboratory, Centre for Quantum Computation & Intelligent Systems, Faculty of Engineering and Information Technology, University of Technology Sydney, Australia*
[b]*Industrial and Systems Engineering Laboratory, School of Mechanical Engineering, Beijing Institute of Technology, China*

E-mails: Jialin.Han@student.uts.edu.cn (J. Han), Guangquan.Zhang@uts.edu.au (G. Zhang), hyg@bit.edu.cn (Y. Hu), Jie.Lu@uts.edu.au (J. Lu).

∗Corresponding author at: Faculty of Engineering and Information Technology, University of Technology Sydney, PO Box 123, Broadway, NSW 2007, Australia. Tel.: +61-2-95141838.

Abstract: Multilevel (including bi-level and tri-level) programming aims to solve decentralized decision-making problems that feature interactive decision entities distributed throughout a hierarchical organization. Since the multilevel programming problem is strongly NP-hard and traditional exact algorithmic approaches lack efficiency, heuristics-based particle swarm optimization (PSO) algorithms have been used to generate an alternative for solving such problems. However, the existing PSO algorithms are limited to solving linear or small-scale bi-level programming problems. This paper first develops a novel bi-level PSO algorithm to solve general bi-level programs involving nonlinear and large-scale problems. It then proposes a tri-level PSO algorithm for handling tri-level programming problems that are more challenging than bi-level programs and have not been well solved by existing algorithms. For the sake of exploring the algorithms' performance, the proposed bi/tri-level PSO algorithms are applied to solve 62 benchmark problems and 810 large-scale problems which are randomly constructed. The computational results and comparison with other algorithms clearly illustrate the effectiveness of the proposed PSO algorithms in solving bi-level and tri-level programming problems.

Keywords: Bi-level programming, tri-level programming, multilevel decision-making, particle swarm optimization, computational intelligence.

1. Introduction

Multilevel programming (also known as multilevel decision-making) attempts to address compromises between interactive decision entities that are distributed throughout a hierarchy[1]. Decision entities at the upper level and the lower level of a multilevel programming problem are respectively termed the *leader* and the *follower* [8, 26]. The leader and follower make their individual decisions in sequence with the aim of optimizing their respective objectives, which means that the follower reacts after and in full knowledge of the leader's decision. However, the leader's decision is implicitly affected by the follower's reaction. Bi-level and tri-level programming problems are two special, typical and popular situations of multilevel programming, which have motivated a number of significant efforts in decision models, solution approaches and applications in areas of mathematics, computer science and business [13, 25, 44].

To achieve a quick understanding of multilevel programming, a tri-level programming case in relation to three-echelon supply chain management can be taken as an example. The three-stage supply chain is composed of a manufacturer, a distributor and a vendor, which are distributed throughout three hierarchical levels. Within a stable sales cycle, they have to hold a certain amount of inventory to fully satisfy market demand, which indicates that one decision entity has to increase its holding inventory if the others reduce their inventories; all of them however seek to minimize their individual inventory holding costs. When inventory decision-making, the manufacturer (the top-level leader) take the lead in developing an optimal inventory plan which considers the current market demand and implicit reactions of other decision entities. According to the decision given by the manufacturer, the distributor (the middle-level follower) then makes an optimal inventory plan; likewise, it references the implicit reaction of the vendor. Lastly, the vendor (the bottom-level follower) determines its inventory to minimize its own cost in light of the fixed inventory plans by the manufacturer and distributor. The decision process will not stop until each decision entity is unwilling to change its decision. This example describes a tri-level programming problem, in which decision entities make decisions in sequence from the manufacturer to the distributor and then to the vendor, but the upper-level decision is affected by implicit reactions of the lower level.

Although multilevel programming problems have been proved to be strongly NP-hard by Ben-Aved and Blair [10] and Bard [7], a number of methodologies have been developed to solve bi-level and tri-level programming problems. In terms of bi-level programming problems, various exact algorithmic approaches have been developed that can be mainly classified into three categories: the vertex enumeration approach [8, 11, 31, 46], the Kuhn-Tucker approach [8, 9, 20, 32], and the penalty function approach [8, 41]. In addition, many certain algorithms have been developed to solve some special kinds of bi-level problems, such as exact penalty method [3], disjunctive cuts method [5] and parametric programming algorithm [16]. While the majority of studies on multilevel programming have focused on bi-level versions, tri-level programming has increasingly attracted investigations into solution approaches since it can be applied to handle many real-world problems [19, 27]. In line with related discussion on the optimality conditions and related geometric properties [6, 29, 42], a range of solution approaches have been developed for solving tri-level programming problems, e.g. cutting plane algorithm [6], penalty function approach [42], Kuhn-Tucker transformation methods [2, 35], multi-parametric programming approach [17], tri-level *K*th-Best algorithm [45] and a category of fuzzy programming approaches [4, 24, 28, 33, 36, 37]. It is notable that readers can refer to the latest survey papers [25, 43] for systematically reviews of up-to-date bi-level and tri-level programming research.

In general, a number of approaches have been proposed to solve bi-level and tri-level programming problems, but these existing approaches are limited to solving linear problems or very time consuming for solving nonlinear and large-scale problems. Nowadays, large-scale and nonlinear features have increasingly appear in multilevel programming problems. For example, business firms usually work in a decentralized manner in a complex supply chain network, then high-dimensional decision variables and nonlinear objectives/constraints are often involved when handling related multilevel programming problems. Consequently, further investigation into solving nonlinear and large-scale multilevel programming problems is necessary.

Particle swarm optimization (PSO) is a population-based heuristic algorithm first proposed by Kennedy and Eberhart [22], which is inspired by the social behavior of organisms such as fish schooling and bird flocking. As PSO is computationally inexpensive in terms of both memory requirements and speed [15], it has a good convergence performance and has been successfully applied in many fields. In relation to solving multilevel programs, Kuo and Huang [23] developed a PSO algorithm for solving linear bi-level programming problems in which decision entities from different decision levels share constraint conditions. Wan et al. [39] proposed a hybrid intelligent algorithm by combining the PSO with chaos searching technique (CST) for solving nonlinear bi-level programming problems. Gao et al. [18] and Zhang et al. [47] respectively applied PSO algorithms to solve bi-level programming models in relation to the pricing problem in supply chain and competitive strategic bidding optimization in electricity markets. Although PSO-based algorithms have been developed to solve multilevel programs, it is only limited to some special (e.g. linear formulations and certain applications) and small-scale bi-level programming problems.

The main contributions of this paper are twofold. First, this paper develops a novel bi-level PSO algorithm to solve general bi-level programs involving nonlinear and large-scale problems. Second, it proposes a tri-level PSO algorithm for solving complicated tri-level programming problems. For the sake of exploring the algorithms' performance, the proposed bi/tri-level PSO algorithms are applied to solve 62 benchmark problems and 810 randomly constructed large-scale problems. Moreover, we compare the computational results with those obtained by the existing heuristic algorithms, such as the existing PSO-CST algorithm [39], evolutionary algorithms [34, 38, 40] and genetic algorithm [12].

This paper is organized as follows. Following the introduction, we present a general bi-level programming problem and develop a bi-level PSO algorithm in Section 2. In Section 3, we present a tri-level PSO algorithm to solve tri-level programming problems. In Section 4, the proposed bi/tri-level PSO algorithms are applied to solve 62 benchmark problems and 810 large-scale problems. Lastly, concluding remarks and further avenues of study are given in Section 5.

2. Bi-level PSO algorithm

In this section, we first propose a general bi-level programming problem and related solution concepts. Second, we develop a bi-level PSO algorithm for solving the proposed bi-level programming problem.

2.1. General bi-level programming problem and solution concepts

The general bi-level programming problem presented by Bard [8] is defined as follows.

Definition 1 [8] For $x \in X \subset R^p$, $y \in Y \subset R^q$, a general bi-level programming problem is defined as:

$$\min_{x \in X} F(x,y) \qquad \text{(1st level, leader)} \tag{1a}$$

$$\text{s.t. } G(x,y) \le 0, \tag{1b}$$

where, for each *x* given by the leader, *y* solves the follower's problem (1c-1d)

$$\min_{y \in Y} f(x,y) \qquad \text{(2nd level, follower)} \tag{1c}$$

$$\text{s.t. } g(x,y) \le 0, \tag{1d}$$

where *x*, *y* are the decision variables of the first level and the second level respectively;

$F, f : R^p \times R^q \to R^1$ are the objective functions of the first level and the second level respectively; $G : R^p \times R^q \to R^m$, $g : R^p \times R^q \to R^n$ are the constraint conditions of the first level and the second level respectively.

To find an optimal solution for the bi-level programming problem (1), relevant solution concepts are presented as follows.

Definition 2 [8]

(1) The constraint region of the bi-level programming problem:

$$S = \{(x, y) \in X \times Y : G(x, y) \le 0, g(x, y) \le 0\}.$$

(2) The feasible set of the second level for each fixed *x*:

$$S(x) = \{y \in Y : g(x, y) \le 0\}.$$

(3) The rational reaction set of the second level:

$$P(x) = \{y \in Y : y \in \arg\min[f(x, y) : y \in S(x)]\}.$$

(4) The inducible region of the bi-level programming problem:

$$IR = \{(x, y) : (x, y) \in S, y \in P(x)\}.$$

(5) The optimal solution set of the bi-level programming problem:

$$OS = \{(x, y) : (x, y) \in \arg\min[F(x, y) : (x, y) \in IR]\}.$$

It is clear from Definition 2 that the constraint domain associated with a bi-level programming problem is implicitly determined by two optimization problems which must be solved in a predetermined sequence from the first level to the second level [21]. To ensure the bi-level programming problem is well posed, the following assumptions based on Definition 2 are commonly made.

Assumption 1. *F*, *f*, *G* and *g* are continuous functions, while *f* and *g* are continuously differentiable.

Assumption 2. *f* is strictly convex in *y* for $y \in S(x)$ where *S*(*x*) is a compact convex set.

Assumption 3. *F* is continuous convex in *x* and *y*.

Under Assumptions 1 and 2, the rational reaction set of the second level *P*(*x*) is a point-to-point map and closed. This implies that the *IR* is compact. Thus, under the assumption 3 solving the bi-level programming problem (1) is equivalent to optimizing the leader's continuous function *F* over the compact set *IR*. It is well known that the solution to such a problem is guaranteed to exist. We thereby develop a bi-level PSO algorithm for the purpose of finding a solution for the bi-level programming problem (1).

2.2. The bi-level PSO algorithm description

PSO is a category of the population-based heuristic algorithm that is motivated by the social behavior of organisms such as fish schooling and bird flocking. The population of PSO is known as a swarm, while each element in the swarm is termed a particle. In a swarm with the size *N*, the position vector of each particle with index *i* $(i=1,2,\ldots,N)$ is denoted as $X_i^t=(x_i^t,y_i^t)$ at iteration *t*, which represents a potential solution to the problem (1). For the sake of convenient discussion, we let $X_i^t=(x_i^t,y_i^t)=(x_{i1}^t,x_{i2}^t)$. At iteration *t*, each particle *i* moves from X_i^t to X_i^{t+1} in the search space at a velocity $V_i^{t+1}=(v_{i1}^{t+1},v_{i2}^{t+1})$ along each dimension. Each particle keeps track of its coordinates in hyperspace which are associated with the best solution (fitness), called *pbest* ($p_i=(p_{i1},p_{i2})$), it has achieved so far; while the PSO algorithm is divided into two versions, respectively known as the GBEST version and the LBEST version, due to different definitions of the best solution [15]. In the GBEST version, the particle swarm optimizer keeps track of the overall best value, called *gbest* ($p_g=(p_{g1},p_{g2})$), and its location obtained thus far by any particle in the population, known as the global neighborhood. For the LBEST version, particles only contain their own and their nearest array neighbors' best information within a local topological neighborhood, rather than that of the entire group. However, in either PSO version, the PSO concept , at each iteration, always consists of an aggregated acceleration of each particle towards its *pbest* and *gbest* position. In this paper, the GBEST version of PSO is followed, and in this section, detailed procedures for solving the problem (1) are developed based on Definition 2.

(1) Initial population

In an initial population of particles with the number *N*, each particle *i* $(i=1,2,\ldots,N)$ can be represented as $X_i^0=(x_i^0,y_i^0)=(x_{i1}^0,x_{i2}^0)$. As an initial population is randomly constructed for the PSO algorithm, we propose a random method to construct an initial population with the size *N*.

First, we randomly generate the required number of the first level decision variables x_i^0 $(i=1,2,\ldots,N)$. Second, we adopt the existing simplex method or interior point method to solve the second level problem $\min\limits_{y\in Y}\{f(x,y):g(x,y)\le 0\}$ under $x=x_i^0$ and obtain the corresponding solution y_i^0. In this way, we complete the construction of the initial population and $X_i^0=(x_i^0,y_i^0)=(x_{i1}^0,x_{i2}^0)$.

Nevertheless, a number of particles of the initial population may occur outside the constraint region *S* particularly in relation to solving large-scale problems with complex constraints. To ensure many more particles of the initial population occur over the constraint region, we propose another construction method to supplement part particles to the initial population.

First, we obtain two solutions $X^{\min}=(x^{\min},y^{\min})$ and $X^{\max}=(x^{\max},y^{\max})$ respectively for solving the problems $\min\{F(x,y):(x,y)\in S\}$ and $\max\{F(x,y):(x,y)\in S\}$.

Second, a formula is defined to construct the initial population:

$(x_i^0,y_i^0)=(x^{\min},y^{\min})+[(x^{\max}-x^{\min})*r_1,(y^{\max}-y^{\min})*r_2]$, where $i=1,2,\ldots,N$, r_1 and r_2 are random numbers uniformly distributed between 0 and 1.

The second method provides more particles occurring over the constraint region, even though the particles may be not uniformly distributed throughout the constraint region. Consequently, when the PSO algorithm is performed for solving small-scale problems, we can only use the first method to construct the initial population; whereas both methods mentioned are able to be combined to construct the initial population for solving large-scale problems. Moreover, the percentage of the population generated by the second method should goes up with the increase in the problem size. Although some particles of the initial population still occur outside the constraint region *S* using these above construction methods, the particles will be tugged to return towards the constraint region *S* at the following iterations if there exist better solutions in *S* [15]; this is an advantage of the PSO algorithm in constructing the initial population.

(2) The updating rules of particles

In the PSO algorithm, each particle *i* moves toward $X_i^{t+1}=(x_i^{t+1},y_i^{t+1})=(x_{i1}^{t+1},x_{i2}^{t+1})$ in the search space at a velocity $V_i^{t+1}=(v_{i1}^{t+1},v_{i2}^{t+1})$ at each iteration *t*. In this paper, the velocity and position of each particle *i* are updated as follows for $j=1,2,i=1,2,\ldots,N$ based on related definitions proposed by Shi and Eberhart [30]:

$$v_{ij}^{t+1}=wv_{ij}^t+c_1r_1(p_{ij}^t-x_{ij}^t)+c_2r_2(p_{gj}^t-x_{ij}^t), \tag{2}$$

$$x_{ij}^{t+1} = x_{ij}^{t} + v_{ij}^{t+1} . \tag{3}$$

We now determine the selection of parameters involved in the formula (2). For the updating velocity, there are usually maximum and minimum velocity levels $v_{\max}$ and $v_{\min}$. If the current velocity $v_{ij}^{t+1} > v_{\max}$, we set $v_{ij}^{t+1} = v_{\max}$; while $v_{ij}^{t+1} = v_{\min}$ if $v_{ij}^{t+1} < v_{\min}$. In the beginning, we set $v_{ij}^{0} = v_{\max}$.

w is the inertia weight, which controls the impact of the previous velocities on the current velocity. The inclusion of the inertia weight involves two definitions proposed by Shi and Eberhart [30]: a fixed constant and a decreasing function with time. In our PSO algorithm, we use the latter to define the inertia weight, because large inertial weight can be used to possess more exploitation ability at the beginning to find a good seed while it is reduced for better local exploitation later on in the search [30]. The inertia weight is represented as:

$$w = w_{\max} - \frac{w_{\max} - w_{\min}}{Iter\_\max} * t , \tag{4}$$

where $w_{\max}$ and $w_{\min}$ are the upper and lower bounds on the inertia weight, which are determined by the practical problem; *Iter_max* is the maximum number of PSO iterations while *t* represents the current iteration number.

c_1 and c_2 are known as learning factors or acceleration coefficients, which control the maximum step size that the particle can do. A recommended choice for constant c_1 and c_2 is integer 2 as proposed by Kennedy and Eberhart [22].

r_1 and r_2 are uniform random numbers between 0 and 1.

(3) Fitness evaluation

For each particle *i* at the iteration *t* $X_i^t = (x_i^t, y_i^t)$, adopt the existing simplex method or interior point method to solve the problem $\min_{y \in Y}\{ f(x, y) : g(x, y) \le 0\}$ under $x = x_i^t$ and obtain the solution (x_i^t, y^*) where $y^* \in P(x_i^t)$. If the solution $(x_i^t, y^*) \in S$, update $X_i^t = (x_i^t, y_i^t) = (x_i^t, y^*)$. Note that $y^* \in P(x_i^t)$ and $(x_i^t, y^*) \in S$ mean $(x_i^t, y^*) \in IR$ by Definition 2, that is, (x_i^t, y^*) is a feasible solution for the bi-level problem (1). The *pbest* solution is $p_i = (p_{i1}, p_{i2}) = (x_i^t, y_i^t)$ if $F(x_i^t, y_i^t) < F(p_{i1}, p_{i2})$ or $f(x_i^t, y_i^t) < f(p_{i1}, p_{i2})$ under $F(x_i^t, y_i^t) = F(p_{i1}, p_{i2})$ where we set

$p_i = (p_{i1}, p_{i2}) = (x_i^0, y_i^0)$ and $F(x_i^0, y_i^0) = +\infty$ at the beginning. The global best solution *gbest* of the swarm at the iteration *t* is $p_g = (p_{g1}, p_{g2})$ where $F(p_{g1}, p_{g2}) = \min\{F(p_{i1}, p_{i2}), i = 1,2,\ldots,N\}$.

(4) Termination criterion

The PSO algorithm will be terminated after a maximum number of iterations *Iter_max* or when it achieves a maximum CPU time.

(5) Computational procedures of the bi-level PSO algorithm

Based on the theoretical basis proposed above, we will present the complete computational procedures of the bi-level PSO algorithm for solving the bi-level programming problem (1).

[Begin]

Step 1: *Initialization*.

(a) Construct the population size *N* and generate the initial population of particles $X_i^0 = (x_i^0, y_i^0), i = 1,2,\ldots,N$;

(b) Initialize the *pbest* solution as $p_i = (p_{i1}, p_{i2}) = (x_i^0, y_i^0)$ and the fitness $F(x_i^0, y_i^0) = +\infty$;

(c) Set the maximum and minimum velocity levels $v_{\max}$ and $v_{\min}$, and initialize $v_{ij}^0 = v_{\max}$;

(d) Set the upper and lower bounds on the inertia weight $w_{\max}$ and $w_{\min}$, acceleration coefficients c_1 and c_2, and the maximum iteration number *Iter_max*;

(e) Set the current iteration number *t*=0 and go to Step 2.

Step 2: *Compute the fitness value and update the pbest solution for each particle.* Set *i*=1 and go to Step 2.1.

Step 2.1: Under $x = x_i^t$, solve the problem $\min_{y \in Y}\{f(x,y) : g(x,y) \le 0\}$ and obtain the solution (x_i^t, y^*). Go to Step 2.2.

Step 2.2: If the solution $(x_i^t, y^*) \in S$, update $X_i^t = (x_i^t, y_i^t) = (x_i^t, y^*)$; otherwise, set $F(x_i^t, y_i^t) = +\infty$. Go to Step 2.3.

Step 2.3: If $F(x_i^t, y_i^t) < F(p_{i1}, p_{i2})$ or $f(x_i^t, y_i^t) < f(p_{i1}, p_{i2})$ under $F(x_i^t, y_i^t) = F(p_{i1}, p_{i2})$, $p_i = (p_{i1}, p_{i2}) = (x_i^t, y_i^t)$. If *i*<*N*, set *i*=*i*+1 and go to Step 2.1; otherwise, go to Step 3.

Step 3: *Update the gbest solution.* Set $p_g = (p_{g1}, p_{g2})$ where

$F(p_{g1}, p_{g2}) = \min\{F(p_{i1}, p_{i2}), i = 1,2,\ldots,N\}$. Go to Step 4.

Step 4: *Termination criterion.* If *t<Iter_max,* go to Step 5; otherwise, stop and $p_g = (p_{g1}, p_{g2})$ is a solution for the bi-level programming problem (1).

Step 5: *Update the inertia weight, and the velocity and the position of each particle* by the formulas (2), (3) and (4). If the current velocity $v_{ij}^{t+1} > v_{\max}$, set $v_{ij}^{t+1} = v_{\max}$; while $v_{ij}^{t+1} = v_{\min}$ if $v_{ij}^{t+1} < v_{\min}$. Set *t*=*t*+1 and go to Step 2.

[End]

3. Tri-level PSO algorithm

In this section, based on the proposed bi-level PSO algorithm, we propose a tri-level PSO algorithm for solving tri-level programming problems.

3.1. General tri-level programming problem and related theoretical properties

The general tri-level programming problem presented by Faísca et al. [17] is defined as follows.

Definition 3 [17] For $x \in X \subset R^p$, $y \in Y \subset R^q$, $z \in Z \subset R^r$, a general tri-level programming problem is defined as:

$$\min_{x \in X} f_1(x, y, z) \quad \text{(1st level, leader)} \tag{5a}$$

$$\text{s.t.} \quad g_1(x, y, z) \le 0, \tag{5b}$$

where, for each *x* given by the 1st level, (*y*, *z*) solves the problems (5c-5f):

$$\min_{y \in Y} f_2(x, y, z) \quad \text{(2nd level, middle-level follower)} \tag{5c}$$

$$\text{s.t.} \quad g_2(x, y, z) \le 0, \tag{5d}$$

where, for each (*x, y*) given by the 1st and 2nd levels, *z* solves the problem (5e-5f) :

$$\min_{z \in Z} f_3(x, y, z) \quad \text{(3rd level, bottom-level follower)} \tag{5e}$$

$$\text{s.t.} \quad g_3(x, y, z) \le 0, \tag{5f}$$

where *x, y, z* are the decision variables of the three levels respectively; $f_1, f_2, f_3 : R^p \times R^q \times R^r \to R$ are the objective functions of the three levels respectively; $g_i : R^p \times R^q \times R^r \to R^{k_i}, i = 1,2,3$ are the constraint conditions of the three levels respectively.

To find an optimal solution for the tri-level programming problem (5), relevant solution concepts are proposed as follows based on the nested hierarchical structure of multilevel programming and the existing research on bi-level programming.

Definition 4 [17]

(1) The constraint region of the tri-level programming problem:

$$S = \{(x, y, z) \in X \times Y \times Z : g_i(x, y, z) \le 0, i = 1,2,3\}.$$

(2) The feasible set of the second level for each fixed *x*:

$$S(x) = \{(y, z) \in Y \times Z : g_2(x, y, z) \le 0, g_3(x, y, z) \le 0\}.$$

(3) The feasible set of the third level for each fixed (*x*, *y*):

$$S(x, y) = \{z \in Z : g_3(x, y, z) \le 0\}.$$

(4) The rational reaction set of the third level:

$$P(x, y) = \{z \in Z : z \in \arg\min[f_3(x, y, z) : z \in S(x, y)\ ]\ .$$

(5) The rational reaction set of the second level:

$$P(x) = \{(y, z) \in Y \times Z : (y, z) \in \arg\min[\ f_2(x, y, z) : (y, z) \in S(x), z \in P(x, y)\ ]\ .$$

(6) The inducible region of the tri-level programming problem:

$$IR = \{(x, y, z) : (x, y, z) \in S, (y, z) \in P(x)\}.$$

(7) The optimal solution set of the tri-level programming problem:

$$OS = \{(x, y, z) : (x, y, z) \in \arg\min[\ f_1(x, y, z) : (x, y, z) \in IR]\}.$$

For the sake of developing an efficient algorithm to solve the tri-level programming problem (5), we now turn our attention to the geometry of the solution space and related theoretical properties. To ensure the problem (5) is well posed, it is common to make the following assumptions based on Definition 4.

Assumption 4. f_1, f_2, f_3, g_1, g_2 and g_3 are continuous functions, whereas f_2, f_3, g_2 and g_3 are continuously differentiable.

Assumption 5. f_3 is strictly convex in *z* for $z \in S(x, y)$ where *S*(*x*, *y*) is a compact convex set, while f_2 is strictly convex in (*y*, *z*) for $(y, z) \in S(x)$ where *S*(*x*) is a compact convex set.

Assumption 6. f_1 is continuous convex in *x*, *y*, and *z*.

Under the assumptions 4 and 5, the rational reaction sets of the third level and the second level *P*(*x*, *y*) and *P*(*x*) are point-to-point maps and closed, which implies that *IR* is compact. Thus, under the assumption 6 solving the tri-level programming problem (5) is equivalent to optimizing the leader's continuous function f_1 over the compact set *IR*. It is well known that the solution to such a problem is guaranteed to exist.

It is noticeable that, if the third-level problem is a convex parametric programming problem that satisfies the Manasarian-Fromowitz constraint qualification (MFCQ) for each fixed (*x*, *y*) [8, 14], the third-level problem is equivalent to the following Kuhn-Tucker conditions (6-9):

$$\nabla_z L(x,y,z,u) = \nabla_z f_3(x,y,z) + u\nabla_z g_3(x,y,z), \tag{6}$$

$$ug_3(x,y,z) = 0, \tag{7}$$

$$g_3(x,y,z) \le 0, \tag{8}$$

$$u \ge 0, \tag{9}$$

where $L(x,y,z,u) = f_3(x,y,z) + ug_3(x,y,z)$ is the Lagrangian function of the third level, $\nabla_z L(x,y,z,u)$ denotes the gradient of the function $L(x,y,z,u)$ with respect to *z*, and *u* is the vector of Lagrangian multipliers.

Theorem 1 [14] A necessary and sufficient condition that $(y,z) \in P(x)$ is that the row vector u exists such that (*x*, *y*, *z*, *u*) satisfies the Kuhn-Tucker conditions (6-9).

Theorem 2 (*x*, *y*, *z*) solves the tri-level programming problem (5) if and only if (*x*, *y*, *z*, *u*) solves the bi-level programming problem (10).

Proof. Based on Theorem 1, the tri-level programming problem (5) is equivalent to solving the bi-level programming problem (10) by replacing the third-level problem with the Kuhn-Tucker conditions (6-9).

$$\min_x f_1(x,y,z) \qquad \text{(1st level, leader)} \tag{10a}$$

$$\text{s.t.}\ \ g_1(x,y,z) \le 0, \tag{10b}$$

where, for each given *x*, (*y*, *z*, *u*) solves (10c-10h)

$$\min_{y,z,u} f_2(x,y,z) \qquad \text{(2nd level, follower)} \tag{10c}$$

$$\text{s.t.}\ \ g_2(x,y,z) \le 0, \tag{10d}$$

$$\nabla_z f_3(x,y,z) + u\nabla_z g_3(x,y,z) = 0, \tag{10e}$$

$$ug_3(x,y,z) = 0, \tag{10f}$$

$$g_3(x,y,z) \le 0, \tag{10g}$$

$$u \ge 0. \tag{10h}$$

Clearly, if (*y*, *z*, *u*) solves (10c-10h) for each given *x*, $(y,z) \in P(x)$ is obtained in line with Theorem 1. Therefore, solving the tri-level programming problem (5) is equivalent to finding a solution (x, y, z, u) to the bi-level programming problem (10) that (x, y, z) is a solution to problem (5). The proof is completed. □

In this study, we extend the bi-level PSO algorithm to a tri-level PSO algorithm for finding a solution (x, y, z) for the tri-level programming problem (5) based on Theorems 1 and 2.

3.2. The tri-level PSO algorithm description

In a swarm with the size N, the position vector of each particle with index i $(i=1,2,\ldots,N)$ is denoted as $X_i^t=(x_i^t,y_i^t,z_i^t)$ at iteration t, which represents a potential solution to the problem (5). For the sake of accessibility, we let $X_i^t=(x_i^t,y_i^t,z_i^t)=(x_{i1}^t,x_{i2}^t,x_{i3}^t)$. At iteration t, each particle i moves from X_i^t to X_i^{t+1} in the search space at a velocity $V_i^{t+1}=(v_{i1}^{t+1},v_{i2}^{t+1},v_{i3}^{t+1})$ along each dimension. Also, we set the *pbest* solution $p_i=(p_{i1},p_{i2},p_{i3})$ and *gbest* solution $p_g=(p_{g1},p_{g2},p_{g3})$. Based on Theorems 1 and 2, the tri-level PSO algorithm for solving the tri-level programming problem (5) is developed in this section.

(1) Initial population

The method of constructing the initial population is similar to the bi-level PSO algorithm. First, we randomly generate the required number of the first level decision variables x_i^0 $(i=1,2,\ldots,N)$. Second, we solve the following problem (11) under $x=x_i^0$ using the branch and bound algorithm [8] or interior point method and obtain the corresponding solution (y_i^0,z_i^0,u_i^0). In this way, we complete the construction of the initial population and $X_i^0=(x_i^0,y_i^0,z_i^0)=(x_{i1}^0,x_{i2}^0,x_{i3}^0)$, $i=1,2,\ldots,N$.

$$\min_{y,z,u} f_2(x,y,z) \tag{11a}$$

$$\text{s.t.}\quad g_2(x,y,z)\le 0, \tag{11b}$$

$$\nabla_z f_3(x,y,z)+u\nabla_z g_3(x,y,z)=0, \tag{11c}$$

$$ug_3(x,y,z)=0, \tag{11d}$$

$$g_3(x,y,z)\le 0, \tag{11e}$$

$$u\ge 0. \tag{11f}$$

(2) The updating rules of particles

Within the tri-level PSO algorithm, each particle i moves toward $X_i^{t+1} = (x_i^{t+1}, y_i^{t+1}, z_i^{t+1}) = (x_{i1}^{t+1}, x_{i2}^{t+1}, x_{i3}^{t+1})$ in the search space at a velocity $V_i^{t+1} = (v_{i1}^{t+1}, v_{i2}^{t+1}, v_{i3}^{t+1})$ at each iteration *t*. The velocity and position of each particle *i* are updated as well as the bi-level PSO algorithm developed in Section 2 by the formulas (2), (3) and (4).

(3) Fitness evaluation

For each particle *i* at the iteration *t* $X_i^t = (x_i^t, y_i^t, z_i^t)$, solve the problem (11) under $x = x_i^t$ using the branch and bound algorithm [8] or interior point method and obtain the solution (x_i^t, y^*, z^*, u^*). If the solution $(x_i^t, y^*, z^*) \in S$, update $X_i^t = (x_i^t, y_i^t, z_i^t) = (x_i^t, y^*, z^*)$. The *pbest* solution is $p_i = (p_{i1}, p_{i2}, p_{i3}) = (x_i^t, y_i^t, z_i^t)$, if $f_1(x_i^t, y_i^t, z_i^t) \le f_1(p_{i1}, p_{i2}, p_{i3})$ where we set $p_i = (p_{i1}, p_{i2}, p_{i3}) = (x_i^0, y_i^0, z_i^0)$ and $f_1(x_i^0, y_i^0, z_i^0) = +\infty$ at the beginning. The global best solution *gbest* of the swarm at the iteration *t* is $p_g = (p_{g1}, p_{g2}, p_{g3})$ where $f_1(p_{g1}, p_{g2}, p_{g3}) = \min\{f_1(p_{i1}, p_{i2}, p_{i3}), i = 1,2,\ldots,N\}$.

(4) Termination criterion

The tri-level PSO algorithm will be terminated after a maximum number of iterations *Iter_max* or when it achieves a maximum CPU time.

(5) Computational procedures of the tri-level PSO algorithm

Based on the bi-level PSO algorithm and the theoretical basis proposed above, we will present the complete computational procedures of the tri-level PSO algorithm for solving the tri-level programming problem (5).

[Begin]

Step 1: *Initialization.*

(a) Construct the population size N and generate the initial population of particles $X_i^0 = (x_i^0, y_i^0, z_i^0), i = 1,2,\ldots,N$ by solving the problem (11);

(b) Initialize the *pbest* solution as $p_i = (p_{i1}, p_{i2}, p_{i3}) = (x_i^0, y_i^0, z_i^0)$ and the fitness $f_1(x_i^0, y_i^0, z_i^0) = +\infty$;

(c) Set the maximum and minimum velocity levels $v_{\max}$ and $v_{\min}$, and initialize $v_{ij}^0 = v_{\max}$;

(d) Set the upper and lower bounds on the inertia weight $w_{\max}$ and $w_{\min}$, acceleration coefficients c_1 and c_2, and the maximum iteration number *Iter_max*;

(e) Set the current iteration number *t*=0 and go to Step 2.

Step 2: *Compute the fitness value and update the pbest solution for each particle.* Set *i*=1 and go to Step 2.1.

Step 2.1: Under $x = x_i^t$, solve the problem (11) using the branch and bound algorithm or interior point method and obtain the solution (x_i^t, y^*, z^*, u^*). Go to Step 2.2.

Step 2.2: If the solution $(x_i^t, y^*, z^*) \in S$, update $X_i^t = (x_i^t, y_i^t, z_i^t) = (x_i^t, y^*, z^*)$; otherwise, set $f_1(x_i^t, y_i^t, z_i^t) = +\infty$. Go to Step 2.3.

Step 2.3: If $f_1(x_i^t, y_i^t, z_i^t) \le f_1(p_{i1}, p_{i2}, p_{i3})$, update $p_i = (p_{i1}, p_{i2}, p_{i3}) = (x_i^t, y_i^t, z_i^t)$. If *i*<*N*, set *i*=*i*+1 and go to Step 2.1; otherwise, go to Step 3.

Step 3: *Update the gbest solution.* Set $p_g = (p_{g1}, p_{g2}, p_{g3})$ where $f_1(p_{g1}, p_{g2}, p_{g3}) = \min\{f_1(p_{i1}, p_{i2}, p_{i3}), i = 1,2,\ldots,N\}$. Go to Step 4.

Step 4: *Termination criterion.* If *t*<*Iter_max*, go to Step 5; otherwise, stop and $p_g = (p_{g1}, p_{g2}, p_{g3})$ is a solution for the tri-level programming problem (5).

Step 5: *Update the inertia weight, and the velocity and the position of each particle* by the formulas (2), (3) and (4) for $j = 1,2,3, i = 1,2,\ldots,N$. If the current velocity $v_{ij}^{t+1} > v_{\max}$, set $v_{ij}^{t+1} = v_{\max}$; while $v_{ij}^{t+1} = v_{\min}$ if $v_{ij}^{t+1} < v_{\min}$. Set *t*=*t*+1 and go to Step 2.

[End]

In view of the procedures of bi/tri-level PSO algorithms, the differences and improvements of the proposed PSO algorithms compared with existing heuristic algorithms can be summarized as follows.

(1) In regard to a nonlinear and large-scale bi-level problem, either of the problems at both levels often appears very difficult to be solved. Whereas the existing bi-level heuristic algorithms [12, 23, 38-40] transform the bi-level problem into a much more complex single-level problem, it will be much easier and more convenient to solve the bi-level problem using the proposed bi-level

PSO algorithm in which the leader's problem and the follower's problem are separated and respectively solved in sequence.

(2) In contrast to the existing bi-level heuristic algorithms [12, 18, 23, 34, 38-40, 47] that cannot be used to solve tri-level problems, the proposed bi-level PSO algorithm can be extended to a tri-level PSO algorithm for solving tri-level problems.

(3) To handle the complexity of the constraint region of nonlinear and large-scale problems, two construction methods of the initial population are given, which can effectively ensure many more particles of the initial population occur over the constraint region and improve the convergence speed of the bi/tri-level PSO algorithms compared with the existing bi-level heuristic algorithms [12, 18, 23, 34, 38-40, 47].

(4) Different from the existing bi-level PSO algorithms [18, 23, 39, 47] using the constant inertia weight, the decreasing inertia weight with time is used to control the velocity of particles in the search space at different stages, which aims to improve both search and convergence abilities of the bi/tri-level PSO algorithms.

We will explore the performance of the proposed bi/tri-level PSO algorithms and demonstrate these aforementioned improvements in the following Section.

4. Computational analysis

A completed computational study is conducted to analyze the performance of the proposed bi/tri-level PSO algorithms. First, we apply the bi/tri-level PSO algorithms to solve 25 bi-level and 8 tri-level benchmark problems involving linear and nonlinear versions. Second, the bi-level PSO algorithm is applied to solve 29 large-scale nonlinear bi-level benchmark problems. Lastly, for the sake of exploring the algorithm performance in depth, we generate 810 large-scale bi-level programming problems using the random method proposed by Calvete et al. [12]. These computational experiments are operated in MATLAB(2014a) programs performed on a 3.47GHz Inter Xeon W3690 CPU with 12G of RAM under a Red Hat Enterprise Linux Workstation. Also, these large-scale problems are randomly generated using the MATLAB(2014a) environment.

4.1. Small-scale benchmark problems

In this section, the bi/tri-level PSO algorithms are applied to solve 25 bi-level and 8 tri-level programming problems involving linear and nonlinear versions. Moreover, we compare the computational results respectively obtained by the bi/tri-level PSO algorithms and other algorithms. The benchmark problems and their related sources are listed in Table 1.

To solve the benchmark problems 1-33, related parameters involved in the bi/tri-level PSO algorithms are chosen in Table 2. Under the parameters in Table 2, the PSO algorithms are performed in 20 independent runs on each of the above 33 benchmark problems. The computational results for bi-level programming problems 1-25 are shown in Table 3.

Table 1. Benchmark problems and their related sources

Problems	Sources
1-14	Problems 1-14 in [39]
15	Ex 1. in [38]
16	Ex 3. in [38]
17	Ex 5. in [38]
18	Ex 7. in [38]
19-20	Problems 1-2 in [40]
21-25	Problems 5-9 in [40]
26	Example 1 in [6]
27	The tri-level numerical illustration in [2]
	Example 1 in [35]
	The tri-level example in [24]
	Example 3 in [33]
28	Example 2 in [35]
	Example 1 in [37]
	Example 1 in [36]
	Example 1 in [28]
29	Example 2 in [36]
30	Example 4.1 in [29]
	Illustrative example 1 in [17]
31	Example 4.2 in [29]
32	The numerical example in [45]
33	The case study in [45]

In Table 3, the solution and the corresponding objective values obtained by the bi-level PSO algorithm are respectively denoted by (x^*, y^*) and (F^*, f^*), while the values obtained by other algorithms are respectively denoted by $(\bar{x}, \bar{y})$ and $(\bar{F}, \bar{f})$. It can be seen from Table 3 that for problems 4, 6-8, 15-16, 18-22 and 24, the solutions obtained by our bi-level PSO algorithm are equal or extremely close to those found by the PSO-CST algorithm [39] and the evolutionary algorithm in [40]. In terms of problems 1-3, 5, 9-14, 17, 23 and 25, the solutions obtained by our bi-level PSO algorithm are better or much better than those found by the compared algorithms in [38, 40], which are highlighted in Table 3. In particular for problems 9-11, 23 and 25, the objective values of the first level respectively obtained by our bi-level PSO algorithm and the compared algorithm are extremely close to one another under different solutions, which implies that there exist multiple solutions for problems 9-11, 23 and 25. Under this situation, using our bi-level PSO algorithm can achieve better or much better objective values for the second level than the compared PSO-CST algorithm and evolutionary algorithm.

Table 2. Parameters employed in the bi/tri-level PSO algorithms for solving problems 1-33

Problems	N	v_{max}	v_{min}	w_{max}	w_{min}	c_1	c_2	***Iter_max***
1	30	1.0	-1.0	0.5	0.01	2.0	2.0	100
2	30	1.0	-1.0	0.5	0.01	2.0	2.0	150
3	20	1.0	-1.0	0.5	0.01	2.0	2.0	60
4	50	1.0	-1.0	0.5	0.01	2.0	2.0	100
5	30	1.0	-1.0	0.5	0.01	2.0	2.0	60
6	50	1.0	-1.0	1.0	0.01	2.0	2.0	150
7	30	1.0	-1.0	0.5	0.01	2.0	2.0	60
8	30	1.0	-1.0	0.5	0.01	2.0	2.0	60
9	60	0.5	-0.5	0.5	0.01	2.0	2.0	100
10	60	1.0	-1.0	0.5	0.01	2.0	2.0	80
11	60	1.0	-1.0	0.5	0.01	2.0	2.0	80
12	40	1.0	-1.0	0.5	0.01	2.0	2.0	60
13	40	1.0	-1.0	0.5	0.01	2.0	2.0	60
14	40	1.0	-1.0	0.5	0.01	2.0	2.0	60
15	50	1.0	-1.0	0.5	0.01	2.0	2.0	150
16	80	1.0	-1.0	1.0	0.01	2.0	2.0	100
17	20	1.0	-1.0	0.5	0.01	2.0	2.0	50
18	30	1.0	-1.0	0.5	0.01	2.0	2.0	60
19	20	1.0	-1.0	0.5	0.01	2.0	2.0	60
20	30	1.0	-1.0	0.5	0.01	2.0	2.0	60
21	50	1.0	-1.0	1.0	0.01	2.0	2.0	150
22	60	0.5	-0.5	0.5	0.01	2.0	2.0	100
23	40	1.0	-1.0	0.5	0.01	2.0	2.0	60
24	40	1.0	-1.0	0.5	0.01	2.0	2.0	60
25	40	1.0	-1.0	0.5	0.01	2.0	2.0	60
26	20	1.0	-1.0	1.0	0.01	2.0	2.0	30
27	20	1.0	-1.0	1.0	0.01	2.0	2.0	30
28	30	1.0	-1.0	1.0	0.01	2.0	2.0	40
29	30	1.5	-1.5	1.0	0.01	2.0	2.0	40
30	20	1.0	-1.0	0.5	0.01	2.0	2.0	30
31	20	1.0	-1.0	0.5	0.01	2.0	2.0	20
32	30	2.0	-2.0	1.0	0.01	2.0	2.0	40
33	30	3.0	-3.0	1.0	0.01	2.0	2.0	40

In relation to problems 2 and 12, it seems in Table 3 that the solutions found by our bi-level PSO algorithm are worse than those obtained by the compared algorithm. With respect to problem 2, the second level will choose y=(y_1, y_2, y_3)=(0, 0, 0) to achieve an optimal objective value $f = 0.4832$ (better than $\bar{f} = 2.3641$) in view of x=(x_1, x_2)=(0.1324, 0.1754). Clearly, the solution

$\bar{y}=(\bar{y}_1,\bar{y}_2,\bar{y}_3)=(0.6935,0.7327,0.2273)$ given by the PSO-CST algorithm in [39] occurs outside the rational reaction set *P*(*x*), which implies that $(\bar{x},\bar{y})=(0.8606,1.4599,0.3188)$ is not a feasible solution for problem 2 according to Definition 2 in Section 2.1. Similarly, the solution $(\bar{x},\bar{y})=(0.8606,1.4599,0.3188)$ given by the evolutionary algorithm in [40] is not a feasible solution for problem 12, because the second level will choose $y=(y_1, y_2)=(1.5382, 0.2166)$ to achieve an optimal objective value $f=2.4980$ (better than $\bar{f}=2.5621$) in view of x=0.8606. Clearly, the algorithms in [39, 40] cannot find an optimal solution for problems 2 and 12. Therefore, our bi-level PSO algorithm performance better than the compared algorithms in [39, 40] in terms of solving problems 2 and 12.

Table 3. The computational results for bi-level programming problems 1-25

Problems	(x^*, y^*)	(F^*, f^*)	$(\bar{x}, \bar{y})$	$(\bar{F}, \bar{f})$
1	(0, 2, 1.875, 0.9063)	(-18.6787, -1.0156)	(0.3844, 1.6124, 1.8690, 0.8041)	(-14.7772, -0.2316)
2	(0, 0.9, 0, 0.6, 0.4)	(-29.2, 3.2)	(0.1324, 0.1754, 0.6935, 0.7327, 0.2273)	(-29.2064, 2.3641)
3	(0, 1, 0)	(1000, 1)	(0.1511, 0.6256, 0.369)	(640.7139, 0.9946)
4	(9.9998, 9.9998)	(99.996, 0)	(10.0020, 9.9961)	(100.0393, 0)
5	(2.0345, 0.8838, 0)	(-1.2312, 7.7818)	(1.8602, 0.9073, 0.005)	(-1.1660, 7.4441)
6	(7.0696, 7.0696, 6.9279, 6.9278)	(1.98, -1.98)	(7.0321, 6.84204, 5.9071, 6.8312)	(1.9816, -1.9816)
7	(20.0282, 14.8381, 0.0282, -5.1619)	(0, 0)	(17.5039, 29.8906, -2.4994, 9.8894)	(0.0527, 0)
8	(17.8377, 20.1712, -2.1623, 0.1712)	(0, 0)	(12.4124, 19.3109, -7.5859, -0.6899)	(0.0004, 0)
9	(20, 5, 10, 5)	(0, 100)	(17.2024, 7.4665, 7.2189, 2.4251)	(0.0075, 125.0854)
10	(10.9317, 9.6004, 10, 9.6004)	(0, 0.868)	(0.1946, 14.9870, 6.1019, 7.9628)	(0, 84.2367)
11	(6.4462, 11.9941, 6.4462, 10)	(0, 3.9763)	(10.6084, 10.0550, 9.4545, 5.1257)	(0.0001, 25.6292)
12	(1.8888, 0.889, 0)	(0, 7.6167)	(0.8606, 1.4599, 0.3138)	(0.0082, 2.5621)
13	(0.6648, 1.5746, 0.0722)	(0, 2.5)	(0.9099, 1.5294, 0.1762)	(0.0374, 2.6969)
14	(0.6648, 1.5746, 0.0722)	(0, 2.5)	(0.9233, 1.5083, 0.1899)	(0.0337, 2.7442)
15	(4, 15, 9.2, 2)	(41.2, -9.2)	(4.000517, 14.999931, 9.199862, 2)	(41.199207, -9.198828)
16	(0, 30, -10, 10)	(0, 100)	(0, 30, -10, 10)	(0, 100)
17	(1, 0)	(1, 0)	(10, 0)	(82, 0)
18	(0, 30, -10, 10)	(0, 100)	(0, 30, -10, 10)	(0, 100)
19	(20, 5, 10, 5)	(225, 100)	(20, 5, 10, 5)	(225, 100)
20	(0, 30, -10, 10)	(0, 100)	(0, 30, -10, 10)	(0, 100)
21	(1.0312, 3.0978, 2.597, 1.7929)	(-8.9172, -6.136)	(1.03, 3.097, 2.59, 1.79)	(-8.92, -6.14)
22	(0.281, 0.4754, 2.3437, 1.0328)	(-7.5774, -0.5777)	(0.27, 0.49, 2.34, 1.036)	(-7.58, -0.574)
23	(38.0907, 60.5204, 2.9985, 2.9985)	(-11.9985, -219.2618)	(12.47, 67.511, 2.999, 2.999)	(-11.999, -163.42)
24	(2, 0, 2, 0)	(-3.6, -2)	(2, -2.84e-8, 2, 0)	(-3.6, -2)
25	(-0.4009, 0.8023, 1.9998, 0)	(-3.9194, -2.0109)	(-0.381, 0.8095, 2, 0)	(-3.92, -2)

Table 4 reports the computational results for tri-level programming problems 26-33. The solution and the corresponding objective values obtained by the tri-level PSO algorithm are respectively denoted by (x^*, y^*, z^*) and (f_1^*, f_2^*, f_3^*), while the values obtained by other solution approaches are denoted by $(\bar{x},\bar{y},\bar{z})$ and $(\bar{f}_1,\bar{f}_2,\bar{f}_3)$. Table 4 clearly shows that the tri-level PSO algorithm can find the same solutions as the compared approaches or much better solutions highlighted in gray color. Note $(\bar{x},\bar{y}^*,\bar{z}^*)$ that $(\bar{y}^*,\bar{z}^*)$ denotes the best reactions of the 2nd level and the 3rd level in the light of $\bar{x}$ determined by the 1st level. According to Definition 4, $(\bar{x},\bar{y},\bar{z})=(\bar{x},\bar{y}^*,\bar{z}^*)$ under $(\bar{x},\bar{y},\bar{z})\in S$ means the solution $(\bar{x},\bar{y},\bar{z})\in IR$, which implies that $(\bar{x},\bar{y},\bar{z})$ is a feasible solution for the tri-level programming problem; otherwise, $(\bar{x},\bar{y},\bar{z})$ is not a feasible solution. Thus, it can be seen from Table 4 that the solutions $(\bar{x},\bar{y},\bar{z})$ in [6] for problem 26, in [24] for problem 27, in [28, 36, 37] for problem 28, in [17] for problem 30 and in [45] for problem 33 are not feasible solutions. In addition, although the solutions $(\bar{x},\bar{y},\bar{z})$ in [2, 33] for problem 27, in [28, 35] for problem 28 and in [36] for problem 29 occurs over *IR*, they can be only considered as local optimal solutions since our tri-level PSO algorithm can find much better solutions for such problems. Thus, the tri-level PSO algorithm provides a better way to solve tri-level programming problems.

Table 4. The computational results for tri-level programming problems 26-33

Problems	(x^*,y^*,z^*)	(f_1^*,f_2^*,f_3^*)	$(\bar{x},\bar{y},\bar{z})$	$(\bar{f}_1,\bar{f}_2,\bar{f}_3)$	$(\bar{x},\bar{y}^*,\bar{z}^*)$
26	(6.6667, 8, 0)	(-10.6667, -8, 0)	(4.6667, 1, 0) [6]	(-16.6667, -1, 0) [6]	(4.6667, 6.5, 4.5)
27	(1.5, 0, 0.5)	(8.5, 0, 0.5)	(0.5, 1, 0.5) [2]	(4.5,1,0.5) [2]	(0.5, 1, 0.5)
			(1.5, 0, 0.5) [35]	(8.5, 0, 0.5) [35]	(1.5, 0, 0.5)
			(1.66, 1, 0.34) [24]	(13.26, 1, 0.34) [24]	No solution
			(0.92, 0.58, 0.5) [33]	(6.18, 0.58, 0.5) [33]	(0.92, 0.58, 0.5)
28	(2.3329, 0.0006,0.3335, 0)	(14.9979, 1.0012, 5.0)	(0.86, 1.86, 0, 0.71) [35]	(13, 4.7, 4.29) [35]	(0.86, 1.86, 0, 0.71)
			(1.59,1.08,0.62,0.06) [36, 37]	(12.01,3.18,4.94) [36, 37]	(1.59, 1.08, 0.705, 0)
			(1.106, 1.525, 0, 0.631) [28]	(13.58, 4.05, 4.37) [28]	(1.106, 1.525, 0.581, 0.244)
			(0.857, 1.857, 0, 0.714) [28]	(13, 4.71, 4.28) [28]	(0.857, 1.857, 0, 0.714)
29	(1, 2, 0, 2, 0)	(14, 2, 8)	(2, 1.99, 1.004, 0, 0.009) [36]	(12.964,5.001,10.188) [36]	(2, 1.99, 1.01, 0, 0.02)
30	(0.5, 1, 1)	(4.5, -2, 1)	(0.5, 1, 1) [29]	(4.5, -2, 1) [29]	(0.5, 1, 1)
			(1, 0.5, 1) [17]	(5, -2, 1) [17]	(1, 1, 0.5)
31	$(x^* \le 2,0,0)$	(0, 0, 0)	$(\bar{x} \le 2,0,0)$ [29]	(0, 0, 0) [29]	$(\bar{x} \le 2,0,0)$
32	(4, 6, 0)	(-20, 10, -8)	(4, 6, 0) [45]	(-20, 10, -8) [45]	(4, 6, 0)
33	No solution	---	(10, 28.33, 11.66) [45]	(146.6667,176.6,343.3) [45]	Unbounded solution

4.2. Large-scale benchmark problems

In this section, we apply the bi-level PSO algorithm to solve the large-scale nonlinear bi-level programming problems 34-62. The sources of the benchmark problems 34-57 are the problems

SMD1-SMD12 with five and 10 dimensions constructed by Sinha et al. [34], while the problems 58-62 with 20 dimensions are cited from the problems (Exs.12-16) solved in [38].

When solving the problems 34-57, the population size and iteration number are chosen as *N*=30, *Iter_max*=60 and *N*=50, *Iter_max*=100 respectively for solving five-dimensional and 10-dimensional problems. The other parameters in the bi-level PSO algorithm are chosen as follows: $v_{\max}$=1.0, $v_{\min}$=-1.0, c_1=c_2=2, $w_{\max}$=0.5, $w_{\min}$=0.01. In response to solving problems 58-62, the related parameters are chosen as $v_{\max}$=0.5, $v_{\min}$=-0.5, c_1=c_2=2, $w_{\max}$=0.5, $w_{\min}$=0.01, *N*=30, *Iter_max*=100.

Table 5. The computational results for five-dimensional test problems 34(SMD1) - 45(SMD12)

Problems	(x^*, y^*)	(F^*, f^*)	$\Delta(F^*, f^*)$	$\Delta(\bar{F}, \bar{f})$
34 (SMD1)	(0, 0, 0, 0, 0)	(0, 0)	(0, 0)	(0.000114, 0.000087)
35 (SMD2)	(-9.1024e-11, 1.3609e-10, -6.4516e-09, 1.0)	(-9.3916e-17, 9.3952e-17)	(9.3916e-17, 9.3952e-17)	(0.000073, 0.000016)
36 (SMD3)	(0, 0, 0, 0, 0)	(0, 0)	(0, 0)	(0.000054, 0.000055)
37 (SMD4)	(-6.3714e-06, 2.7123e-06, -1.2107e-08, 1.3537e-08, 4.1916e-04)	(-1.7330e-07, 1.7339e-07)	(1.7330e-07, 1.7339e-07)	(0.000023, 0.000057)
38 (SMD5)	(-6.0842e-08, -3.3604e-06, 1.0, 1.0, 0.0039)	(-1.2665e-10, 1.3795e-10)	(1.2665e-10, 1.3795e-10)	(0.000002, 0.000009)
39 (SMD6)	(-3.4925e-08, 2.5489e-05, 4.4706e-06, 4.4706e-06, 2.5474e-05)	(6.8968e-10, 1.4570e-15)	(6.8968e-10, 1.4570e-15)	(0.000108, 0.000061)
40 (SMD7)	(3.8445e-09, -1.1977e-11, -6.4516e-09, -6.4516e-09, 1.0)	(-9.3943e-17, 9.3943e-17)	(9.3943e-17, 9.3943e-17)	(0.000016, 0.000177)
41 (SMD8)	(4.2671e-11, 2.4561e-09, 1.0, 1.0, 0.0233)	(8.8549e-12, 2.0450e-10)	(8.8549e-12, 2.0450e-10)	(0.000174, 0.000027)
42 (SMD9)	(3.2473e-05, -2.7497e-07, -1.6235e-04, -1.6235e-04, 5.1978e-04)	(-3.2198e-07, 3.2409e-07)	(3.2198e-07, 3.2409e-07)	(0.000017, 0.000054)
43 (SMD10)	(1.0, 1.0, 1.0, 1.0, 0.7854)	(4.0, 3.0)	(0, 0)	(0.034759, 0.018510)
44 (SMD11)	(8.2462e-06, -6.2919e-04, 1.0379e-07, 1.0379e-07, 2.7166)	(-1.0, 1.0)	(0, 0)	(0.0131643, 0.129893)
45 (SMD12)	(1.0, 1.0, 1.0, 1.0, 0.7849)	(4.9990, 3.0)	(0.001, 0)	(0.032372, 0.000206)

The computational results for the problems 34-45 and problems 46-57 are respectively provided in Tables 5 and 6. In Tables 5 and 6, the solution and the corresponding objective values obtained by the bi-level PSO algorithm are respectively denoted by (x^*, y^*) and (F^*, f^*), while the objective values obtained by the nested bi-level evolutionary algorithm developed in [34] are denoted by $(\bar{F}, \bar{f})$. Let (F, f) be the objective values under the exact solution. $\Delta(F^*, f^*) = (|F^* - F|, |f^* - f|)$ and $\Delta(\bar{F}, \bar{f}) = (|\bar{F} - F|, |\bar{f} - f|)$ are adopted to reflect the accuracy of the solution respectively obtained by both of the algorithms. The smaller number of $\Delta(F^*, f^*)$ and $\Delta(\bar{F}, \bar{f})$ means the higher accuracy of the solution obtained. It can be seen from Tables 5 and

6 that our bi-level PSO algorithm is able to find a more accurate solution than the nested bi-level evolutionary algorithm.

Table 6. The computational results for 10-dimensional test problems 46(SMD1) - 57(SMD12)

Problems	(x^*, y^*)	(F^*, f^*)	$\Delta(F^*, f^*)$	$\Delta(\bar{F}, \bar{f})$
46 (SMD1)	(0, 0, 0, 0, 0, 0, 0, 0, 0, 0)	(0, 0)	(0, 0)	(0.000332, 0.000018)
47 (SMD2)	(1.6899e-08, -3.0972e-08, 8.2156e-08, -2.1181e-07, 2.5143e-07, -6.4507e-09, -6.4507e-09, -6.4507e-09, 1.0, 1.0)	(1.1593e-13, 8.1410e-15)	(1.1593e-13, 8.1410e-15)	(0.000066, 0.000011)
48 (SMD3)	(1.4331e-06, -1.0599e-06, -1.4075e-06, -4.3816e-07, 3.8293e-06, -3.7340e-09, -3.7340e-09, 5.1703e-09, 1.3951e-08, 1.3951e-08)	(2.0014e-11, 5.1590e-12)	(2.0014e-11, 5.1590e-12)	(0.000359, 0.000033)
49 (SMD4)	(2.3924e-07, 4.5629e-08, 3.7434e-07, 1.0472e-06, 1.6518e-07, -1.4097e-08, -1.4097e-08, -1.4097e-08, 3.1005e-04, 3.0963e-04)	(-1.9119e-07, 1.9119e-07)	(1.9119e-07, 1.9119e-07)	(0.000286, 0.000027)
50 (SMD5)	(2.4397e-05, -3.1364e-06, -5.7256e-06, -2.1074e-05, -4.7159e-06, 1.0, 1.0, 1.0, 0.0040, 0.0038)	(9.9401e-10, 7.4795e-10)	(9.9401e-10, 7.4795e-10)	(0.000052, 0.000009)
51 (SMD6)	(-8.7492e-06, -7.0808e-06, 7.6839e-05, 4.9603e-05, 5.0376e-05, 1.6532e-08, 5.3412e-05, 5.3412e-05, 4.9566e-05, 5.0338e-05)	(1.6735e-08, 6.0309e-09)	(1.6735e-08, 6.0309e-09)	(0.001435, 0.000082)
52 (SMD7)	(-1.9409e-09, 1.4642e-08, 1.4642e-08, -7.3262e-09, -7.1216e-09, -6.4688e-09, -6.4688e-09, 1.1635e-04, 1.0, 1.0)	(-2.6032e-08, 1.0577e-16)	(2.6032e-08, 1.0577e-16)	(0.006263, 0.000127)
53 (SMD8)	(2.3591e-07, 4.3256e-05, 1.5413e-06, 1.2043e-07, 2.3549e-06, 1.0, 1.0, 1.0, 0.0320, 0.0324)	(9.9992e-05, 4.5035e-05)	(9.9992e-05, 4.5035e-05)	(0.003122, 0.000157)
54 (SMD9)	(0.0012, 3.6938e-04, 3.2828e-05, -2.9128e-04, -3.1246e-04, -9.4424e-04, -9.0683e-04, -9.3510e-04, 0.0055, -0.0157)	(-2.7495e-04, 2.7829e-04)	(2.7495e-04, 2.7829e-04)	---
55 (SMD10)	(0.50, 0.50, 0.50, 0.50, 0.50, 0.50, 0.50, 0.50, 0.4636, 0.4636)	(12.0, 7.50)	(0, 0)	---
56 (SMD11)	(-1.8430e-06, 4.6479e-08, -6.4905e-07, -3.6628e-07, 1.6103e-06, 6.6159e-08, 6.6159e-08, 6.6159e-08, 2.0281, 2.0281)	(-1.0, 1.0)	(0, 0)	---
57 (SMD12)	---	---	---	---

Table 7 displays the computational results for problems 58-62 with 20 dimensions. As shown in Table 7, the solutions found by our bi-level PSO algorithm are equal to those obtained by the compared algorithm in [38] for problems 58-59 and 62. With regard to problems 60-61, the bi-level PSO algorithm can achieve little better in terms of objective values. As highlighted in Table 7, we can find that problems 60-61 have multiple solutions that can achieve objective values extremely close to each other. To conclude, the results indicate that our bi-level PSO algorithm can find the same solutions as the compared algorithm or better solutions for 20-dimensional nonlinear bi-level problems.

Table 7. The computational results for 20-dimensional test problems 58-62

Problems	(x^*, y^*)	(F^*, f^*)	$(\bar{x}, \bar{y})$	$(\bar{F}, \bar{f})$
58	(1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0)	(0,1)	(1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0)	(0,1)
59	(1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0)	(0,1)	(1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0)	(0,1)
60	(-0.0859,-0.6008,0.2454,-0.1412,-1.2518, -0.1978,-0.9856,-0.1297,-0.7022,-5, 0,0,0,0,0,0,0,0,0,0)	(1.2388e-6,1)	(1.149034,0.08833383,1.254797,1.182997, 2.130051,1.742112,0.3082794, 1.591319,1.409942,-0.2195419, 0,0,0,0,0,0,0,0,0,0)	(4.64e-6,1)
61	(0.8882,0.7552,5,4.3309,-0.5017,0.8485, -1.2183,2.2813,-1.5316,0.6639, 0,0,0,0,0,0,0,0,0,0)	(1.7316e-7,1)	(-1.275612,0.4240169,-1.292204,-0.57017, 1.238698,2.83057,1.313386, 0.65589,-2.799304,-1.467915)	(1.12e-5,1)
62	(1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0)	(0,1)	(1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0)	(0,1)

4.3. Assessing the efficiency performance of the proposed bi-level PSO algorithm

In this section, we aim to assess the efficiency performance of the proposed bi-level PSO algorithm in relation to solve large-scale problems. In Section 4.2, we provide the related parameters employed in the bi-level PSO algorithm for solving large-scale nonlinear benchmark problems. Much less iterations need to be executed by our bi-level PSO algorithm than the evolutionary algorithm [34] that needs 330 and 678 iterations at least respectively for solving five-dimensional and 10-dimensional problems. Clearly, our bi-level PSO algorithm has a better convergence and efficiency performance than the evolutionary algorithm in solving large-scale nonlinear bi-level programming problems. However, the increase in the number of decision variables (e.g. more than 20 dimensions) may result in bi-level problems having no solutions apart from some special versions [34]; thus, there are not sufficient benchmark nonlinear problems in the existing research that can be used to explore the algorithm efficiency. In this study, to explore the algorithm efficiency in solving much larger-scale (e.g. 20 dimensions or much more) problems, we will apply the bi-level PSO algorithm to solve sufficient large-scale (40, 60 and 100 dimensions) linear bi-level problems that can be randomly generated.

Sufficient large-scale linear bi-level programming problems are randomly generated using the method proposed by Calvete et al. [12]. The problems are constructed by the following formulation format:

$$\min_{x\geq 0} F(x,y) = c_1x + d_1y \qquad \text{(1st level)}$$

$$\min_{y\geq 0} f(x,y) = c_2x + d_2y \qquad \text{(2nd level)}$$

$$\text{s.t.} \quad Ax + By \leq b\,.$$

The objective functions' coefficients (c_1, d_1, c_2, d_2) of both level are randomly generated from the uniform distribution on [-10, 10]. For the sake of ensuring the problem is well posed, the coefficients of one constraint condition are chosen from uniform random numbers between 0 and 10, whereas the remainder elements of the coefficient matrix are uniformly distributed between -10 and 10. The right-hand side of each constraint condition is the sum of the absolute value of the coefficients in the constraint condition. According to the construction method by Calvete et al. [12], the test problems are classified into three groups (G_1, G_2 and G_3) by the number n of decision variables of the bi-level programming problem, shown in Table 8. n_1 and n_2 respectively denote the number of decision variables of the first level and the second level, while m denotes the number of constraint conditions of the bi-level problem. It can be seen from Table 8 that there are nine problem types in each test problem group by different combinations of n_1, n_2 and m. In this computational study, we randomly construct 30 test problems within each problem type; thus, there are $30\times 9\times 3 = 810$ bi-level problems randomly generated in total within three test problem groups.

Table 8. 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	3	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

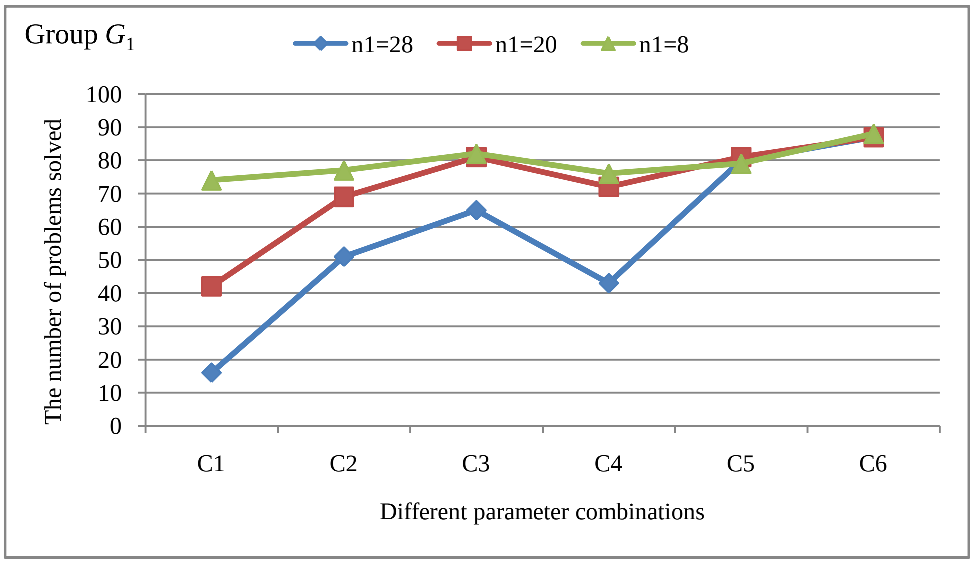
Within the bi-level PSO algorithm, the key parameters involve the inertia weight *w*, the population size *N* and the maximum number of iterations *Iter_max*. To explore the influence of the three parameters on the performance of the bi-level PSO algorithm, each test problem is solved under six kinds of parameter combinations of the bi-level PSO algorithm, which involve C_1

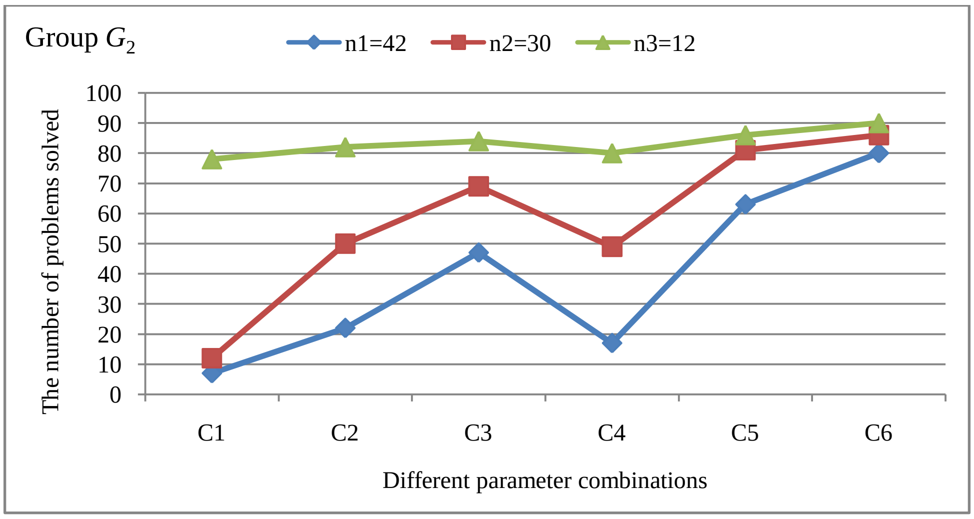
($w_{\max}$=1.0, N=100, *Iter_max*=300), C_2 ($w_{\max}$=0.75, N=100, *Iter_max*=300), C_3 ($w_{\max}$=0.50, N=100, *Iter_max*=300) , C_4 ($w_{\max}$=1.0, N=50, *Iter_max*=500), C_5 ($w_{\max}$=0.75, N=50, *Iter_max*=500) and C_6 ($w_{\max}$=0.50, N=50, *Iter_max*=500). In addition, we set other parameters within the bi-level PSO algorithm: $v_{\max}$=5.0, $v_{\min}$=-5.0, $w_{\min}$=0.01, c_1=c_2=2. For 810 test problems, each of them is carried out 16 runs under each parameter combination. In terms of each test problem, we define $F^{\min}$ as the best objective value of the first level obtained from all parameter combinations; if the best objective value F of the first level obtained from 16 runs under each parameter combination equals to $F^{\min}$, we consider that the bi-level PSO algorithm can find a solution for the test problem under the parameter combination. Table 9 displays the number of test problems successfully solved under each parameter combination.

Table 9. The number of test problems successfully solved under each parameter combination

G_1: n=40							G_2: n=60							G_3: n=100						
n_1-n_2-m	C_1	C_2	C_3	C_4	C_5	C_6	n_1-n_2-m	C_1	C_2	C_3	C_4	C_5	C_6	n_1-n_2-m	C_1	C_2	C_3	C_4	C_5	C_6
28-12-12	5	17	23	18	27	28	42-18-18	3	10	15	7	20	26	70-30-30	8	9	17	8	16	25
28-12-20	3	16	20	13	27	29	42-18-30	2	7	18	6	22	26	70-30-50	10	13	22	6	18	23
28-12-32	8	18	22	12	26	30	42-18-48	2	5	14	4	21	28	70-30-80	11	15	22	9	19	24
20-20-12	13	22	25	21	24	29	30-30-18	5	14	19	17	26	27	50-50-30	1	7	17	7	16	26
20-20-20	13	21	27	23	27	28	30-30-30	2	16	24	17	27	29	50-50-50	2	4	15	7	18	27
20-20-32	16	26	29	28	30	30	30-30-48	5	20	26	15	28	30	50-50-80	6	12	17	11	25	25
8-32-12	19	20	23	20	21	29	12-48-18	23	25	26	22	27	30	20-80-30	10	18	22	14	21	28
8-32-20	26	27	29	27	28	29	12-48-30	28	28	29	28	29	30	20-80-50	10	25	28	18	26	29
8-32-32	29	30	30	29	30	30	12-48-48	27	29	29	30	30	30	20-80-80	13	23	29	21	28	30
Total	132	197	228	191	240	262	Total	97	154	200	146	230	256	Total	71	126	189	101	187	237

Table 9 clearly shows that different combinations of the inertia weight w, the population size N and the maximum number of iterations *Iter_max* have significant influences on the performance of the bi-level PSO algorithm. As shown in Table 9, most test problems are successfully solved under the parameter combination C_6 within each problem group, which means that the bi-level PSO algorithm shows higher performance under C_6 than other parameter combinations. However, the algorithm performance under C_1 - C_5 becomes more and more close to that under C_6 following the decline of the number n_1 of the first level decision variables, in particular in groups G_1 and G_2; Fig.1 clearly presents these results, which display the total number of test problems that have the same number of the first level decision variables successfully solved under C_1 - C_6. Also, it is clear in Fig. 1 that the algorithm performance under each parameter combination experiences a noticeable upward trend along with a decrease in the number n_1 of the first level decision variables within groups G_1 and G_2. In terms of problem group G_3, it is noticeable that the number of test problems with n_1=70 successfully solved exceeds that with n_1=50, which implies that the increase in the population size of the bi-level PSO algorithm is able to improve its performance in solving these problems when much more decision variables of the first level are involved.





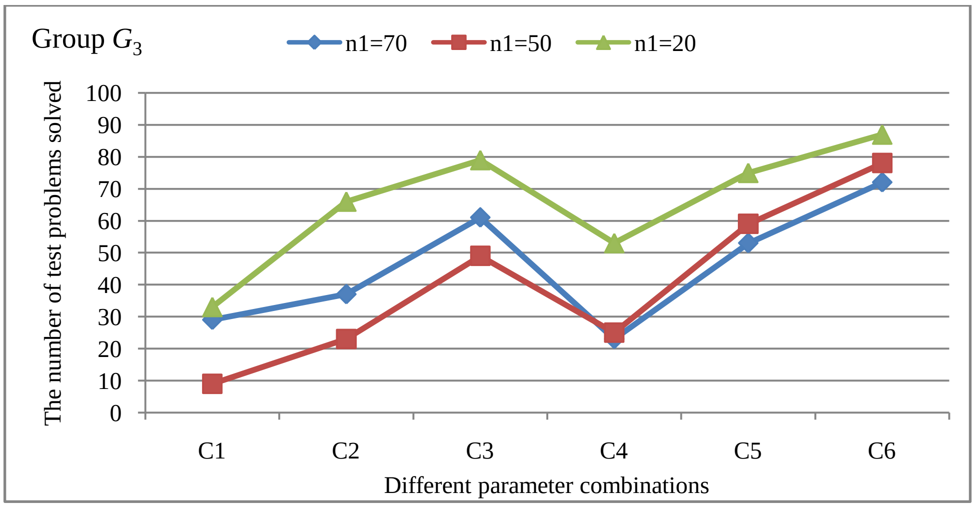
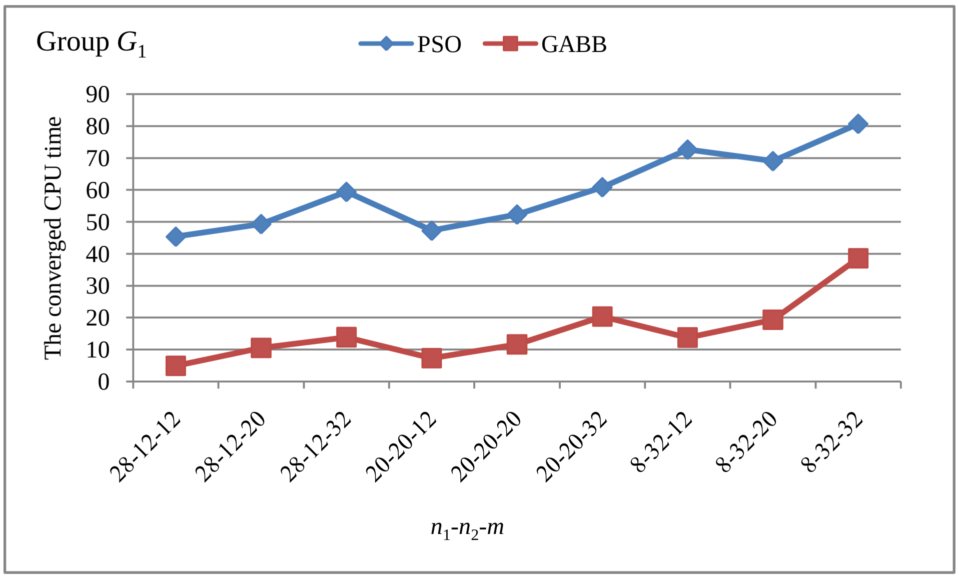


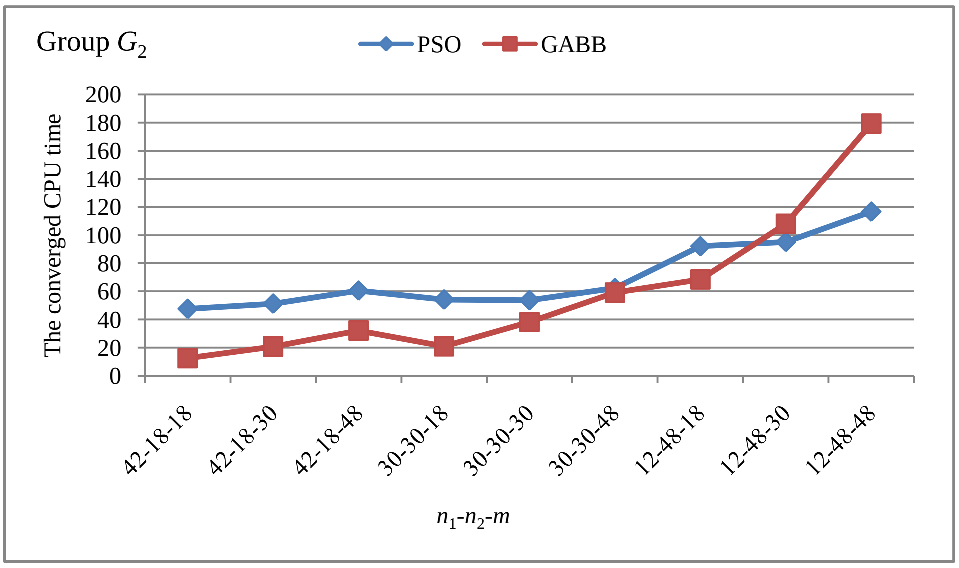
Fig. 1. The performance of the bi-level PSO algorithm following different parameter combinations

To explore more in depth, we compare the algorithm efficiency of our bi-level PSO algorithm with that of the genetic algorithm based on bases (GABB) developed by Calvete et al. [12] for solving these test problems randomly constructed. We examine the convergent CPU time and the total CPU time of all iterations completed for both algorithms. Table 10 shows the average of the convergent CPU time (in seconds) and the total CPU time (in seconds) for each problem type using both algorithms. Note that the computational results of the bi-level PSO algorithm are obtained under the parameter combination C_6, while the GABB is performed under its best related parameter combination presented by Calvete et al. [12].

Table 10. The computational results respectively obtained by the bi-level PSO algorithm and GABB

Test Problems		PSO			GABB	
Group	n_1-n_2-m	Convergent iteration number	Convergent time	Total time	Convergent time	Total time
G_1	28-12-12	356.03	45.34	74.22	4.92	27.32
	28-12-20	376.38	49.28	64.48	10.49	33.65
	28-12-32	344.50	59.35	84.90	13.85	41.26
	20-20-12	360.97	47.21	66.92	7.31	39.25
	20-20-20	365.39	52.28	72.57	11.58	51.36
	20-20-32	353.80	60.82	87.52	20.29	67.35
	8-32-12	265.48	72.58	136.15	13.75	72.10
	8-32-20	240.28	68.97	137.76	19.34	82.65
	8-32-32	237.50	80.64	169.46	38.52	110.65
G_2	42-18-18	365.38	47.54	63.58	12.68	56.38
	42-18-30	370.81	51.27	69.68	20.96	71.84
	42-18-48	397.96	60.49	74.36	32.29	95.43
	30-30-18	366.52	54.18	73.87	21.03	105.57
	30-30-30	368.03	53.69	73.09	38.35	128.64
	30-30-48	383.47	62.36	81.26	59.31	169.91
	12-48-18	351.53	92.03	129.38	68.62	284.05
	12-48-30	358.03	95.11	135.48	108.24	361.54
	12-48-48	324.50	116.66	180.28	179.35	440.72
G_3	70-30-30	318.96	48.83	76.69	54.69	115.20
	70-30-50	328.52	51.32	78.05	83.61	159.61
	70-30-80	337.29	67.33	99.80	145.37	231.49
	50-50-30	368.65	56.69	77.16	134.29	331.85
	50-50-50	340.52	75.70	107.85	221.89	428.96
	50-50-80	344.32	71.54	103.76	410.36	630.87
	20-80-30	339.29	86.84	128.72	771.59	1652.19
	20-80-50	396.31	132.91	168.76	1299.76	2238.47
	20-80-80	427.47	200.41	234.34	1684.53	2489.63





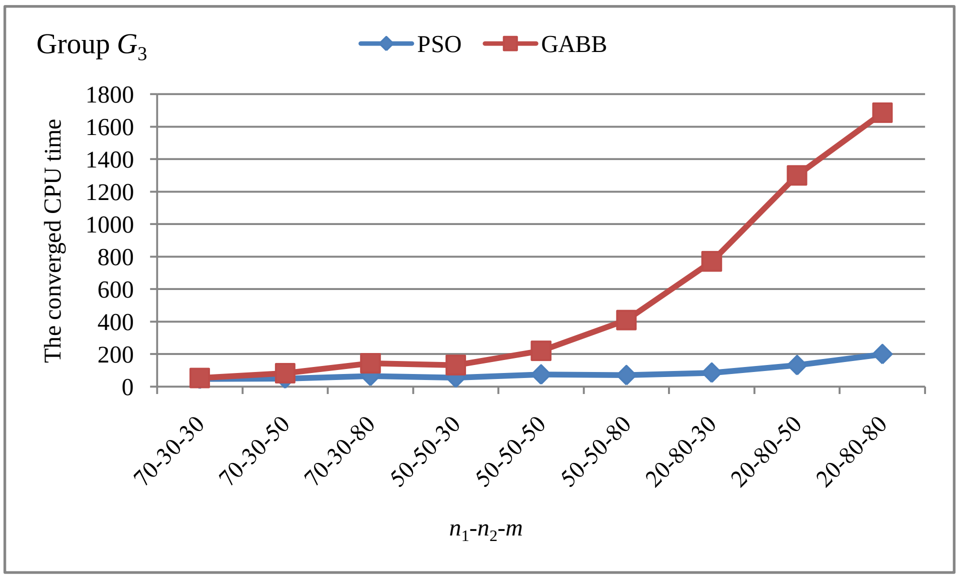


Fig. 2. The average of the convergent CPU time

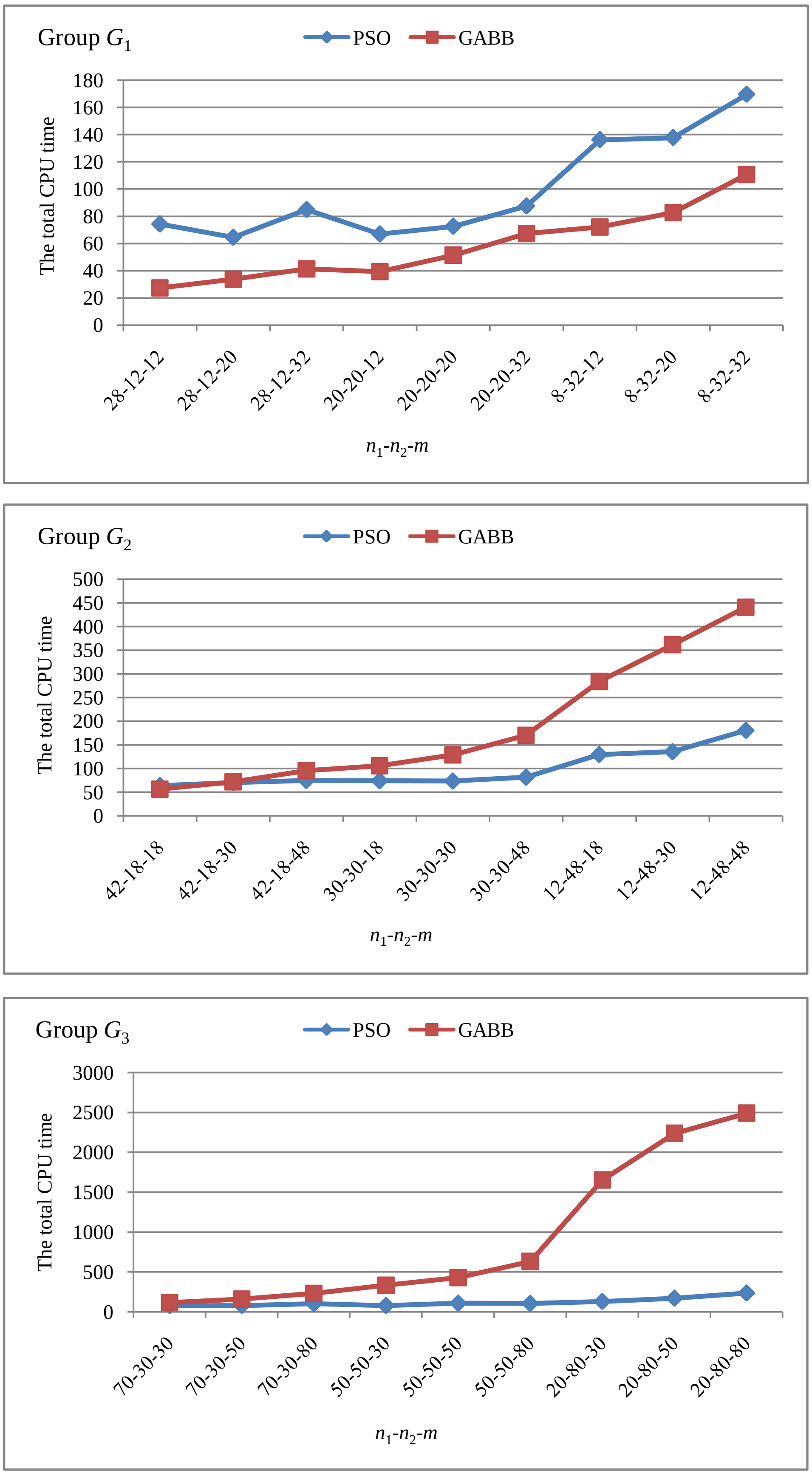


Fig. 3. The average of the total CPU time of all iterations completed

It can be seen from Table 10 that our bi-level PSO algorithm spends more CPU times obtaining the best solution and completing all iterations for solving problem group G_1, however, both CPU times of our bi-level PSO algorithm become less and less than GABB following the increase in the number of decision variables within problem groups G_2 and G_3; Fig. 2 and Fig. 3 display much more evident results. Fig.2 and Fig.3 clearly show that both the convergent and total CPU times of GABB increase steeply with the increase in the size of the test problems. In particular in group G_3, GABB takes much more CPU times than our bi-level PSO algorithm to converge to the best solution and complete all the iterations, which implies that our bi-level PSO algorithm has a significant advantage in solving larger-scale problems.

5. Conclusions and further study

In this paper, we sought to solve bi-level and tri-level programming problems using PSO. To validate and illustrate the effectiveness of the proposed bi/tri-level PSO algorithms, we applied them to solve 62 benchmark problems and 810 large-scale problems which were randomly constructed. We also compared the computational results with those obtained by the existing PSO-CST algorithm [39], evolutionary algorithms [34, 38, 40] and genetic algorithm [12]. On the one hand, the computational results of these benchmark bi-level and tri-level programming problems reported that our bi/tri-level PSO algorithms are able to find much better solutions than the compared algorithms. On the other hand, the computational results of these large-scale problems clearly indicated that our bi-level PSO algorithm shows much better performance in terms of efficiency than the compared algorithms following the problem size becoming larger and larger. In conclusion, the proposed bi-level PSO algorithm provides a practical way to solve nonlinear and large-scale bi-level programming problems; also, it can be extended to a tri-level PSO algorithm for solving tri-level programming problems.

In the future, we will apply the proposed multilevel programming techniques and bi/tri-level PSO algorithms to model and solve decentralized decision-making problems in the real world, e.g. supply chain management, logistics and hierarchical production operations. Moreover, we will turn our attention to the generalization of the proposed bi/tri-level PSO algorithms into problem scenarios with uncertain issues, e.g. multilevel programming problems with fuzzy and/or random parameters.

Acknowledgements

This work is supported by the Australian Research Council (ARC) under discovery grant DP140101366.

References

[1] M.A. Abo-Sinna, I.A. Baky, Interactive balance space approach for solving multi-level multi-objective programming problems, Information Sciences, 177 (2007) 3397-3410.

[2] G. Anandalingam, A mathematical programming model of decentralized multi-level systems, Journal of the Operational Research Society, 39 (1988) 1021-1033.

[3] Z. Ankhili, A. Mansouri, An exact penalty on bilevel programs with linear vector optimization lower level, European Journal of Operational Research, 197 (2009) 36-41.

[4] S.R. Arora, R. Gupta, Interactive fuzzy goal programming approach for bilevel programming problem, European Journal of Operational Research, 194 (2009) 368-376.

[5] C. Audet, J. Haddad, G. Savard, Disjunctive cuts for continuous linear bilevel programming, Optimization Letters, 1 (2007) 259-267.

[6] J.F. Bard, An investigation of the linear three level programming problem, IEEE Transactions on Systems, Man, and Cybernetics, SMC-14 (1984) 711-717.

[7] J.F. Bard, Some properties of the bilevel programming problem, Journal of Optimization Theory and Applications, 68 (1991) 371-378.

[8] J.F. Bard, Practical Bilevel Optimization: Algorithms and Applications, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[9] J.F. Bard, J.E. Falk, An explicit solution to the multi-level programming problem, Computers & Operations Research, 9 (1982) 77-100.

[10] O. Ben-Aved, C.E. Blair, Computational difficulties of bilevel linear programming, Operations Research, 38 (1990) 556-560.

[11] W.F. Bialas, M.H. Karwan, On two-level optimization, IEEE Transactions on Automatic Control, AC-26 (1982) 211-214.

[12] H.I. Calvete, C. Galé, P.M. Mateo, A new approach for solving linear bilevel problems using genetic algorithms, European Journal of Operational Research, 188 (2008) 14-28.

[13] S.-W. Chiou, A bi-level programming for logistics network design with system-optimized flows, Information Sciences, 179 (2009) 2434-2441.

[14] S. Dempe, Foundations of Bilevel Programming, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[15] R. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: Proceedings of the 6th International Symposium on Micro Machine and Human Science, 1995, pp. 39-43.

[16] N.P. Faísca, V. Dua, B. Rustem, P.M. Saraiva, E.N. Pistikopoulos, Parametric global optimisation for bilevel programming, Journal of Global Optimization, 38 (2007) 609-623.

[17] N.P. Faísca, P.M. Saraiva, B. Rustem, E.N. Pistikopoulos, A multi-parametric programming approach for multilevel hierarchical and decentralised optimisation problems, Computational Management Science, 6 (2007) 377-397.

[18] Y. Gao, G. Zhang, J. Lu, H.M. Wee, Particle swarm optimization for bi-level pricing problems in supply chains, Journal of Global Optimization, 51 (2011) 245-254.

[19] J. Han, J. Lu, Y. Hu, G. Zhang, Tri-level decision-making with multiple followers: Model, algorithm and case study, Information Sciences, 311 (2015) 182-204.

[20] P. Hansen, B. Jaumard, G. Savard, New branch-and-bound rules for linear bilevel programming, SIAM Journal on Scientific and Statistical Computing, 13 (1992) 1194-1217.

[21] V. Kalashnikov, R. Ríos-Mercado, A natural gas cash-out problem: A bilevel programming framework and a penalty function method, Optimization and Engineering, 7 (2006) 403-420.

[22] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of The 1995 IEEE International Conference on Neural Networks, 1995, pp. 1942-1948.

[23] R.J. Kuo, C.C. Huang, Application of particle swarm optimization algorithm for solving bi-level linear programming

problem, Computers & Mathematics with Applications, 58 (2009) 678-685.

[24] Y.-J. Lai, Hierarchical optimization: A satisfactory solution, Fuzzy Sets and Systems, 77 (1996) 321-335.

[25] J. Lu, J. Han, Y. Hu, G. Zhang, Multilevel decision-making: A survey, Information Sciences, 346-347 (2016) 463-487.

[26] J. Lu, C. Shi, G. Zhang, On bilevel multi-follower decision making: General framework and solutions, Information Sciences, 176 (2006) 1607-1627.

[27] J. Lu, G. Zhang, J. Montero, L. Garmendia, Multifollower trilevel decision making models and system, IEEE Transactions on Industrial Informatics, 8 (2012) 974-985.

[28] S. Pramanik, T.K. Roy, Fuzzy goal programming approach to multilevel programming problems, European Journal of Operational Research, 176 (2007) 1151-1166.

[29] G.Z. Ruan, S.Y. Wang, Y. Yamamoto, S.S. Zhu, Optimality conditions and geometric properties of a linear multilevel programming problem with dominated objective functions, Journal of Optimization Theory and Applications, 123 (2004) 409-429.

[30] C. Shi, R. Eberhart, A modified particle swarm optimizer, in: Proceedings of The 1998 IEEE International Conference on Evolutionary Computation, 1998, pp. 69-73.

[31] C. Shi, H. Lu, G. Zhang, An extended Kth-best approach for linear bilevel programming, Applied Mathematics and Computation, 164 (2005) 843-855.

[32] C. Shi, J. Lu, G. Zhang, An extended Kuhn-Tucker approach for linear bilevel programming, Applied Mathematics and Computation, 162 (2005) 51-63.

[33] H.-S. Shih, Y.-J. Lai, E.S. Lee, Fuzzy approach for multi-level programming problems, Computers & Operations Research, 23 (1996) 73-91.

[34] A. Sinha, P. Malo, K. Deb, Test problem construction for single-objective bilevel optimization, Evoluation Computation, 22 (2014) 439-477.

[35] S. Sinha, A comment on Anandalingam (1988). A mathematical programming model of decentralized multi-level systems. J Opl Res Soc 39: 1021-1033, Journal of the Operational Research Society, 52 (2001) 594-596.

[36] S. Sinha, Fuzzy mathematical programming applied to multi-level programming problems, Computers & Operations Research, 30 (2003) 1259-1268.

[37] S. Sinha, Fuzzy programming approach to multi-level programming problems, Fuzzy Sets and Systems, 136 (2003) 189-202.

[38] Z. Wan, L. Mao, G. Wang, Estimation of distribution algorithm for a class of nonlinear bilevel programming problems, Information Sciences, 256 (2014) 184-196.

[39] Z. Wan, G. Wang, B. Sun, A hybrid intelligent algorithm by combining particle swarm optimization with chaos searching technique for solving nonlinear bilevel programming problems, Swarm and Evolutionary Computation, 8 (2013) 26-32.

[40] Y. Wang, Y.-C. Jiao, H. Li, 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 (2005) 221-232.

[41] D. White, G. Anandalingam, A penalty function approach for solving bi-Level linear programs, Journal of Global Optimization, 3 (1993) 397-419.

[42] D.J. White, Penalty function approach to linear trilevel programming, Journal of Optimization Theory and Applications, 93 (1997) 183-197.

[43] G. Zhang, J. Han, J. Lu, Fuzzy Bi-level Decision-Making Techniques: A Survey, International Journal of Computational Intelligence Systems, 9 (2016) 25-34.

[44] G. Zhang, J. Lu, Y. Gao, Multi-Level Decision Making: Models, Methods and Applications, Springer, Berlin, 2015.

[45] G. Zhang, J. Lu, J. Montero, Y. Zeng, Model, solution concept and the Kth-best algorithm for linear tri-level programming, Information Sciences, 180 (2010) 481-492.

[46] G. Zhang, C. Shi, J. Lu, An extended Kth-Best approach for referential-uncooperative bilevel multi-follower decision

making, International Journal of Computational Intelligence Systems, 1 (2008) 205-214.
[47] G. Zhang, G. Zhang, Y. Gao, J. Lu, Competitive strategic bidding optimization in electricity markets using bilevel programming and swarm technique, IEEE Transactions on Industrial Electronics, 58 (2011) 2138-2146.