Solving Tri-level Programming Problems Using a Particle Swarm Optimization Algorithm

Jialin Han[1,2], Guangquan Zhang[1], Yaoguang Hu[2], Jie Lu[1]

[1]Centre for Quantum Computation & Intelligent Systems, Faculty of Engineering and Information Technology, University of Technology Sydney, Australia
[2]School of Mechanical Engineering, Beijing Institute of Technology, China

hjl@bit.edu.cn, Guangquan.Zhang@uts.edu.au, hyg@bit.edu.cn, Jie.Lu@uts.edu.au

***Abstract*—Tri-level programming, a special case of multilevel programming, arises to deal with decentralized decision-making problems that feature interacting decision entities distributed throughout three hierarchical levels. As tri-level programming problems are strongly NP-hard and the existing solution approaches lack universality in solving such problems, the purpose of this study is to propose an intelligence-based heuristic algorithm to solve tri-level programming problems involving linear and nonlinear versions. In this paper, we first propose a general tri-level programming problem and discuss related theoretical properties. A particle swarm optimization (PSO) algorithm is then developed to solve the tri-level programming problem. Lastly, a numerical example is adopted to illustrate the effectiveness of the proposed PSO algorithm.**

Keywords—tri-level programming; bi-level programming; computational intelligence; particle swarm optimization; Kuhn-Tucker conditions.

I. INTRODUCTION

Tri-level programming (also known as tri-level decision-making), has been developed to address compromises among interacting decision entities that are distributed throughout three hierarchical levels, which is a subfamily of multilevel programming [1] motivated by Stackelberg game theory [2]. Decision entities at the three hierarchical levels are respectively termed the top-level leader, the middle-level follower and the bottom-level follower [3]. In a tri-level decision-making process, the decision entities make their individual decisions in sequence from the top level to the middle level and then to the bottom level with the aim of optimizing their respective objectives. The decision process means that the higher-level decision entity has priority in making decisions to optimize its own objectives and the lower-level decision entity reacts in view of decisions made by the higher level. However, the decision of each entity is affected by actions of the others. The decision process is repeatedly executed until the Stackelberg equilibrium among them is achieved. This hierarchical decision-making process often appears in many decentralized decision problems in the real world, such as supply chain management [4], resource allocation [5, 6] and hierarchical production operations [7].

Whereas the majority of studies on multi-level programming were focused on bi-level programming, research on tri-level programming problems has attracted increasingly investigations into solution approaches since it can be used to deal with many decentralized decision problems in the real world. Bard [8] first presented an investigation of linear tri-level programming and designed a cutting plane algorithm to solve such problems, based on which White [9] proposed a penalty function approach for linear tri-level programming problems. Anandalingam [10] and Sinha [11] developed Kuhn-Tucker transformation methods to find local optimal solutions for linear tri-level programming problems. Ruan, et al. [12] discussed optimality conditions and related geometric properties of a linear tri-level programming problem with dominated objective functions. Faśca, et al. [13] studied a multi-parametric programming approach to solve tri-level hierarchical and decentralized optimization problems based on parametric global optimization for bi-level programming [14]. Zhang, et al. [3] developed a tri-level *K*th-Best algorithm to solve linear tri-level programming problems. Lai [15], Shih, et al. [16] and Sinha [17, 18] developed fuzzy approaches to find solutions to linear multilevel programming problems involving tri-level programming problems, and further, Pramanik and Roy [19] proposed another fuzzy approach using linear goal programming to solve such problems. However, the existing solution approaches are limited to solving tri-level programming problems in the linear version or in a special situation where all decision entities share the same constraint conditions. In particular, the fuzzy approaches can only solve the tri-level programming problems in which decision entities from different levels prefer to cooperate with one another. In this way, the fuzzy approaches can only used to find some satisfactory solutions rather than the optimal solutions, because the cooperation is inhibited in classical multilevel programming problems. Consequently, further investigation into solution approaches for solving tri-level programming problems is necessary.

Since tri-level programming problems are strongly NP-hard and the existing solution approaches lack universality in solving such problems, intelligent heuristic algorithms may be used to generate an alternative for solving such problems. Particle swarm optimization (PSO) is a population-based heuristic algorithm first proposed by Kennedy and Eberhart [20], which is inspired by the social behavior of organisms such as fish schooling and bird flocking. As PSO requires only primitive mathematical operators, and is computationally

inexpensive in terms of both memory requirements and speed, it has a good convergence performance and has been successfully employed to solve bi-level programming problems [21-23]. In this study, we will try to develop a PSO algorithm to solve tri-level programming problems.

The main contribution of this paper is the provision of a PSO algorithm to solve tri-level programming problems involving linear and nonlinear versions. This paper first presents a general tri-level programming problem and discusses related theoretical properties. It then develops a PSO algorithm based on the Kuhn-Tucker conditions to solve the proposed tri-level programming problem. Lastly, a numerical example is used to illustrate the effectiveness of the proposed PSO algorithm.

II. The Tri-level Programming Problem and Related Theoretical Properties

In this section, we will propose the tri-level programming problem and discuss related theoretical properties.

A. The Tri-level Programming Problem and Solution Concepts

The general tri-level programming problem presented by Fa śca, et al. [13] is defined as follows.

Definition 1 [13] For $x \in X \in R^p$, $y \in Y \in R^q$, $z \in Z \in R^r$, a general tri-level programming problem is defined as:

$$\begin{aligned}
&\min_{x \in X} f_1(x, y, z) && \text{(1st level)}\\
&\text{s.t. } g_1(x, y, z) \le 0,\\
&\quad \text{where } y, z \text{ solve}\\
&\quad \min_{y \in Y} f_2(x, y, z) && \text{(2nd level)}\\
&\quad \text{s.t. } g_2(x, y, z) \le 0,\\
&\qquad \text{where } z \text{ solves}\\
&\qquad \min_{z \in Z} f_3(x, y, z) && \text{(3rd level)}\\
&\qquad \text{s.t. } g_3(x, y, z) \le 0,
\end{aligned} \tag{1}$$

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 (also called a Stackelberg solution) for the tri-level programming problem (1), 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 2

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:

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

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:

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

4) The rational reaction set of the second level:

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

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

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

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

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

B. Related Theoretical Properties

For the sake of developing an efficient algorithm to solve the tri-level programming problem (1), we now turn our attention to the geometry of the solution space and related theoretical properties. To ensure the problem (1) is well posed, it is common to make the following assumptions based on Definition 2.

Assumption 1 $f_1, f_2, f_3, g_1, g_2, g_3$ are continuous functions, whereas f_2, f_3, g_2, g_3 are continuously differentiable.

Assumption 2 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 3 f_1 is continuous convex in x, y, and z.

Under the assumptions 1 and 2, 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 3 solving the tri-level programming problem (1) 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 convex parametric programming problem that satisfies the Manasarian-Fromowitz constraint qualification (MFCQ) for each fixed (x, y) [24, 25], the third-level problem is equivalent to the following Kuhn-Tucker conditions (2-5):

$$\nabla_z L(x, y, z, u) = \nabla_z f_3(x, y, z) + u \nabla_z g_3(x, y, z), \tag{2}$$

$$u g_3(x, y, z) = 0, \tag{3}$$

$$g_3(x, y, z) \le 0, \tag{4}$$

$$u \geq 0, \tag{5}$$

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 [25] A necessary and sufficient condition that $(y, z) \in P(x)$ is that there exists the row vector u such that (x, y, z, u) satisfies the Kuhn-Tucker conditions (2-5).

Based on Theorem 1, the tri-level programming problem (1) can be transformed into the bi-level programming problem (6) by replacing the third-level problem with the Kuhn-Tucker conditions (2-5).

$$\begin{aligned}
&\min_{x} \ f_1(x, y, z) && \text{(1st level)} \\
&\text{s.t. } g_1(x, y, z) \leq 0, \\
&\quad \text{where } y, z \text{ solve} \\
&\quad \min_{y,z,u} \ f_2(x, y, z) && \text{(2nd level)} \\
&\quad \text{s.t. } g_2(x, y, z) \leq 0, \\
&\qquad \nabla_z f_3(x, y, z) + u\nabla_z g_3(x, y, z) = 0, \\
&\qquad ug_3(x, y, z) = 0, \\
&\qquad g_3(x, y, z) \leq 0, \\
&\qquad u \geq 0.
\end{aligned} \tag{6}$$

Therefore, we have the following theorem.

Theorem 2 (x, y, z) solves the tri-level programming problem (1) if and only if (x, y, z, u) solves the bi-level programming problem (6).

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

III. The Particle Swarm Optimization Algorithm

Particle swarm optimization (PSO) is a category of 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 swarm, while each particle in the swarm is termed particle. In a swarm with the size N, the position vector of each particle with index $i (i = 1,2,\cdots,N)$ 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 (1). For the sake of convenient discussion, 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. Each particle keeps track of its coordinates in hyperspace which are associated with the best solution (fitness), called *pbest* ($p_i = (p_{i1}, p_{i2}, p_{i3})$), 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 global best solution [26]. In the GBEST version, the particle swarm optimizer keeps track of the overall best value, called *gbest* ($p_g = (p_{g1}, p_{g2}, p_{g3})$), and its location obtained thus far by any particle in the population, known as the global neighborhood. For the LBEST version, particles have information only of their own and their nearest array neighbors' best within a local topological neighborhood, rather than that of the entire group. However, in either PSO version, the PSO concept always consists of, at each iteration, an aggregated acceleration of each particle towards its *pbest* and *gbest* position. In this paper, the GBEST version of PSO is followed, and detailed procedures for solving the problem (1) will be developed in this section based on Theorems 1 and 2.

1) Initial population

In an initial population of particles with the number N, each particle $i (i = 1,2,\cdots,N)$ can be represented as $X_i^0 = (x_i^0, y_i^0, z_i^0) = (x_{i1}^0, x_{i2}^0, x_{i3}^0)$. 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 $(i = 1,2,\cdots,N)$. Second, we solve the following problem (7) under $x = x_i^0$ using the branch and bound algorithm [24] or interior point method (in Matlab) and obtain the corresponding solution (y_i^0, z_i^0, u_i^0). In this way, we complete the construction of initial population and $X_i^0 = (x_i^0, y_i^0, z_i^0) = (x_{i1}^0, x_{i2}^0, x_{i3}^0)$.

$$\begin{aligned}
&\min_{y,z,u} \ f_2(x, y, z) \\
&\text{s.t. } g_2(x, y, z) \leq 0, \\
&\quad \nabla_z f_3(x, y, z) + u\nabla_z g_3(x, y, z) = 0, \\
&\quad ug_3(x, y, z) = 0, \\
&\quad g_3(x, y, z) \leq 0, \\
&\quad u \geq 0.
\end{aligned} \tag{7}$$

Note that some particles of the initial population may occur outside the constraint region S although the constraint region S is a convex set, but the particles will be tugged to return towards the constraint region S at following iterations if there exist better solutions in S [26]; 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 $X_i^{t+1} = (x_i^{t+1}, y_i^{t+1}, z_i^{t+1}) = (x_{i1}^{t+1}, x_{i2}^{t+1}, x_{i3}^{t+1})$ 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. In this paper, the velocity and position of each particle i are updated as follows for $j = 1,2,3, i = 1,2,\cdots,N$ based on related definitions proposed by Shi and Eberhart [27]:

$$v_{ij}^{t+1} = wv_{ij}^{t} + c_1 r_1 (p_{ij}^{t} - x_{ij}^{t}) + c_2 r_2 (p_{gj}^{t} - x_{ij}^{t}), \quad (8)$$

$$x_{ij}^{t+1} = x_{ij}^{t} + v_{ij}^{t+1}. \quad (9)$$

We now determine the selection of parameters involved in the formula (10). 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 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 [27]: 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 [27]. The inertia weight is represented as:

$$w = w_{\max} - \frac{w_{\max} - w_{\min}}{Iter\_\max} * t, \quad (10)$$

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 [20].

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

3) Fitness evaluation

For each particle *i* at the iteration *t* $X_i^t = (x_i^t, y_i^t, z_i^t)$, solve the problem (7) under $x = x_i^t$ using the branch and bound algorithm [24] or interior point method (in Matlab) 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 $f_1(p_{g1}, p_{g2}, p_{g3}) = \min\{ f_1(p_{i1}, p_{i2}, p_{i3}), i = 1,2,\cdots,N\}$.

4) Termination criterion

The PSO algorithm will be terminated after a maximum number of iterations *Iter_max* or with achieving a maximum CPU time.

5) Computational procedures of the PSO algorithm

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

Step 1: *Initialization*.

a) Construct the population size *N* and generate the initial population of particles $X_i^0 = (x_i^0, y_i^0, z_i^0), i = 1,2,\cdots,N$ by solving the problem (7);

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*=1 and go to Step 2.

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

Step 2.1: Under $x = x_i^t$, solve the problem (7) using the branch and bound algorithm or interior point method (in Matlab) 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 $X_i^t = (x_i^t, y_i^t, z_i^t) = (x_i^t, y^*, z^*)$. Go to Step 2.3.

Step 2.3: If $f_1(x_i^t, y_i^t, z_i^t) \le f_1(p_{i1}, p_{i2}, p_{i3})$, $p_i = (p_{i1}, p_{i2}, p_{i3}) = (x_i^t, y_i^t, z_i^t)$. 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 $f_1(p_{g1}, p_{g2}, p_{g3}) = \min\{ f_1(p_{i1}, p_{i2}, p_{i3}), i = 1,2,\cdots,N\}$. 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 (1).

Step 5: *Update the inertia weight, and the velocity and the position of each particle*, respectively by the formulas (8), (9) and (10). 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.

IV. A Numerical Example

In this section, we will first illustrate how the proposed PSO algorithm works through solving a nonlinear tri-level programming problem.

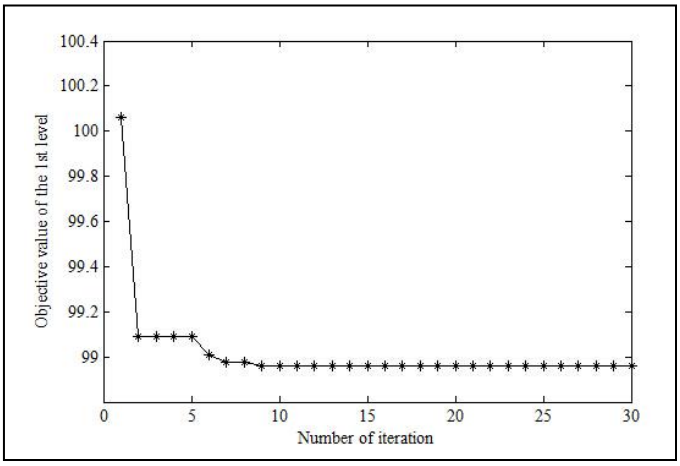

Fig.1. The converged curve of the leader's objective value.

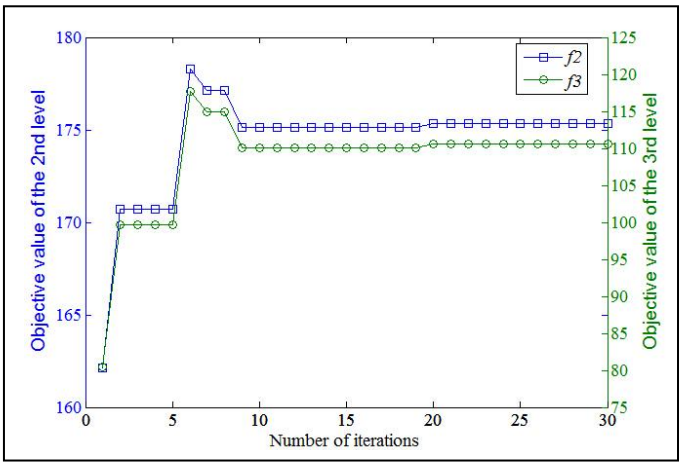

Fig.2. The objective values of the 2nd level (*f2*) and the 3rd level (*f3*).

Consider the following nonlinear tri-level programming problem:

$$\min_x \ f_1(x,y,z) = x^2 + y^2 + z(z-5) \qquad \text{(1st level)}$$

$$\text{s.t. } x - y + z \le 10,$$

$$0 \le x \le 15,$$

where y, z solve

$$\min_{y \in Y} \ f_2(x,y,z) = (x+y)^2 + (z-10)^2 \qquad \text{(2nd level)} \quad (11)$$

$$\text{s.t. } x^2 - (y-20)^2 \le 0,$$

$$-y^2 + z^2 \le 0,$$

$$0 \le y \le 18,$$

where z solves

$$\min_z \ f_3(x,y,z) = x^2 + (y + 2z - 30)^2 \quad \text{(3rd level)}$$

$$\text{s.t. } x + y + z \le 20,$$

$$z^2 - 12z \le 0.$$

TABLE I. PARAMETERS EMPLOYED IN THE PSO ALGORITHM

N	v_{max}	v_{min}	w_{max}	w_{min}	c_1	c_2	*Iter_max*
20	1	-1	1.0	0.01	2.0	2.0	30

While the existing solution approaches cannot be used to solve the nonlinear tri-level programming problem, we will use the proposed PSO algorithm to find a solution for the problem. Based on the PSO procedures developed in Section III, related parameters are initialized in TABLE I.

The computational procedures proposed in Section III are implemented by Matlab R2014a. The experiment results imply that we can obtain a converged solution under the parameters shown in TABLE I. The converged curve of the 1st level (leader)'s objective value $f_1(x,y,z)$ is shown in Fig. 1.

It can be seen from Fig.1 that the leader's objective value has converged to $f_1(x,y,z) = 98.9583$ since the 20th iteration, while the objective values of the 2nd level and the 3rd level converge to $f_2(x,y,z) = 175.3673$ and $f_3(x,y,z) = 110.6383$, as shown in Fig. 2. In this way, we can obtain a converged *gbest* solution $p_g = (p_{g1}, p_{g2}, p_{g3}) = (5.8346, 7.0827, 7.0827)$ for the nonlinear tri-level programming problem (11). Clearly, the PSO algorithm provides a practical way to solve tri-level programming problems.

V. CONCLUSIONS AND FURTHER STUDY

This study developed a PSO-based intelligent algorithm to solve tri-level programming problems. First, we proposed the general tri-level programming problem and discussed related theoretical properties. Second, we presented the procedures of the PSO algorithm, based on the Kuhn-Tucker conditions, for solving tri-level programming problems. Lastly, we illustrated how the PSO algorithm works through a numerical example. The computational results show that the PSO algorithm provides a practical way to solve tri-level programming problems involve linear and nonlinear versions. In the future, we will extend the PSO algorithm to solve large-scale tri-level programming problems in applications and explore the execution efficiency of the algorithm.

Acknowledgment

This work is supported by the Australian Research Council (ARC) under discovery grant DP140101366 and the National High Technology Research and Development Program of China (NO. 2013AA040402).

References

[1] L. Vicente and P. Calamai, "Bilevel and multilevel programming: A bibliography review," Journal of Global Optimization, vol. 5, pp. 291-306, 1994.

[2] H. V. Stackelberg, The Theory of Market Economy. Oxford: Oxford University Press, 1952.

[3] G. Zhang, J. Lu, J. Montero, and Y. Zeng, "Model, Solution concept and the Kth-best algorithm for linear tri-level programming," Information Sciences vol. 180, pp. 481-492, 2010.

[4] X. Xu, Z. Meng, and R. Shen, "A tri-level programming model based on Conditional Value-at-Risk for three-stage supply chain management," Computers & Industrial Engineering, vol. 66, pp. 470-475, 2013.

[5] Y. Yao, T. Edmunds, D. Papageorgiou, and R. Alvarez, "Trilevel optimization in power network defense," IEEE Transactions on Systems, Man, and Cybernetics, vol. 37, pp. 712-718, 2007.

[6] S. Mitiku, "A multilevel programming approach to decentralized (or hierarchical) resource allocation systems," Proceedings in Applied Mathematics and Mechanics, vol. 7, pp. 2060003-2060004, 2007.

[7] S. A. Torabi, M. Ebadian, and R. Tanha, "Fuzzy hierarchical production planning (with a case study)," Fuzzy Sets and Systems, vol. 161, pp. 1511-1529, 2010.

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

[9] D. J. White, "Penalty function approach to linear trilevel programming," Journal of Optimization Theory and Applications, vol. 93, pp. 183-197, 1997.

[10] G. Anandalingam, "A mathematical programming model of decentralized multi-level systems" Journal of Operational Research Society, vol. 39, pp. 1021-1033, 1988.

[11] S. Sinha, "A comment on Anandalingam (1988). A mathematical programming model of decentralized multi-level systems. Journal of Operational Research Society, vol. 52, pp. 594-596, 2001.

[12] G. Z. Ruan, S. Y. Wang, Y. Yamamoto, and S. S. Zhu, "Optimality Conditions and Geometric Properties of a Linear Multilevel Programming Problem with Dominated Objective Functions," Journal of Optimization Theory and Applications, vol. 123, pp. 409-429, 2004.

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

[14] N. Faísca, V. Dua, B. Rustem, P. Saraiva, and E. Pistikopoulos, "Parametric global optimisation for bilevel programming," Journal of Global Optimization, vol. 38, pp. 609-623, 2007.

[15] Y.-J. Lai, "Hierarchical optimization: A satisfactory solution," Fuzzy Sets and Systems, vol. 77, pp. 321-335, 1996.

[16] H.-S. Shih, Y.-J. Lai, and E. S. Lee, "Fuzzy approach for multi-level programming problems," Computers & Operations Research, vol. 23, pp. 73-91, 1996.

[17] S. Sinha, "Fuzzy programming approach to multi-level programming problems," Fuzzy Sets and Systems, vol. 136, pp. 189-202, 2003.

[18] S. Sinha, "Fuzzy mathematical programming applied to multi-level programming problems," Computers & Operations Research, vol. 30, pp. 1259-1268, 2003.

[19] Pramanik and T. K. Roy, "Fuzzy goal programming approach to multilevel programming problems," European Journal of Operational Research, vol. 176, pp. 1151-1166, 2007.

[20] J. Kennedy and R. Eberhart, "Particle swarm optimization," in Proceedings of IEEE International Conference on Neural Networks, vol.4, pp. 1942-1948, 1995.

[21] R. J. Kuo and C. C. Huang, "Application of particle swarm optimization algorithm for solving bi-level linear programming problem," Computers & Mathematics with Applications, vol. 58, pp. 678-685, 2009.

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

[23] T. Zhang, T. Hu, X. Guo, Z. Chen, and Y. Zheng, "Solving high dimensional bilevel multiobjective programming problem using a hybrid particle swarm optimization algorithm with crossover operator," Knowledge-Based Systems, vol. 53, pp. 13-19, 2013.

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

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

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

[27] C. Shi and R. Eberhart, "A modified particle swarm optimizer," IEEE World Congress on Computational Intelligence, pp. 69-73, 1998.