Information Sciences Letters

Volume 6
Issue 1 *Jan. 2017*

Article 1

2021

Solving Linear Bilevel Programming via Particle Swarm Algorithm with Heuristic Pattern Searc

Mohamed A. Tawhid
Department of Mathematics and Statistics, Faculty of Science, Thompson Rivers University, Kamloops, BC, Canada V2C 0C8\\ Department of Mathematics and Computer Science, Faculty of Science, Alexandria University, Moharam Bey 21511, Alexandria, Egypt, Mtawhid@tru.ca

Garrett Paluck
Department of Mathematics, Simon Fraser University Burnaby, British Columbia V5A 1S6 Canada, Mtawhid@tru.ca

Information Sciences Letters
An International Journal

http://dx.doi.org/10.18576/isl/060101

Solving Linear Bilevel Programming via Particle Swarm Algorithm with Heuristic Pattern Search

Mohamed A. Tawhid [1,2,*] *and Garrett Paluck*[3]

[1] Department of Mathematics and Statistics, Faculty of Science, Thompson Rivers University, Kamloops, BC, Canada V2C 0C8
[2] Department of Mathematics and Computer Science, Faculty of Science, Alexandria University, Moharam Bey 21511, Alexandria, Egypt
[3] Department of Mathematics, Simon Fraser University Burnaby, British Columbia V5A 1S6 Canada

Received: 27 Feb. 2016, Revised: 13 Dec. 2016, Accepted: 26 Dec. 2016
Published online: 1 Jan. 2017

Abstract: A metaheuristic approach is proposed for solving linear bilevel programming problem using the Memetic Particle Swarm Algorithm which uses a Heuristic Pattern Search as the local search. The proposed algorithm has proven to be stable and capable of generating the optimal solution to the linear bilevel programming problem. The numerical results show that the metaheuristic approach is both feasible and efficient.

Keywords: Linear bilevel programming, particle swarm algorithm, heuristic pattern search.

1 Introduction

Linear Bilevel Programming (BLP) problems appear in various scientific and engineering applications including resource allocation, finance budget, price control, transportation, and network design. It is defined by an optimization problem, which has a second-order optimization problem as a constraint. The BLP problem has been proved to be NP-hard [5].

In this paper, we follow the Karush-Kuhn-Tucker (KKT) optimality conditions for the second order optimization problem. We simplify the linear BLP problem to a regular linear programming problem with complementary constraints. After preforming a smoothing method, we use a penalty function to evaluate the problem using different PSO based algorithms.

Particle Swarm Optimization (PSO) is a popular stochastic, population-based search algorithm developed in 1995. Its popularity can be attributed to its easy implementation and ability to efficiently solve many variations of problems in science and engineering, including optimal design of power systems [1], feature selection for structure-activity correlations in medical applications [2], biological applications [4], size and shape optimization [6], [18], environmental applications [10], analysis in chemical processes [13], bioinformatics [15], task assignment problems [19], industrial control [14] and numerical optimization [16], [15].

PSO is based off of the patterns of social dynamics and the interactions among the members of organized colonies. PSO is classified as a swarm intelligence algorithm. PSO has many common key characteristics with Evolutionary Algorithms (EAs), such as Genetic Algorithms [7], Evolution Strategies [21] and Differential Evolutions [22], thereby sharing many aspects of their behavior.

The Pattern Search Algorithm (PS) was originally created in 1961 [8]. It was used primarily to find the optimal solutions to nonlinear problems. It could efficiently solve unconstrained, bound constrained, as well as linearly and nonlinearly nonsmooth constrained problems. The Heuristic Pattern Search (HPS) Algorithm is a newer version created in 2011 [20] which adds a random descent direction if a pattern move proves successful. HPS proved to be more efficient than the regular Pattern Search.

We will be using an algorithm that combines PSO with the HPS algorithm as the local search method [20]. Thus, resulting in an efficient Memetic PSO scheme. The results are compared with the corresponding results of the local and global variants of both PSO and the original version of MPSO [17].

* Corresponding author e-mail: Mtawhid@tru.ca

2 Linear Bilevel Programming

2.1 Smoothing Method

The goal of this article is to find the optimal solution of a linear bilevel programming problem. At first, the problem is in a form where we cannot solve it. Therefore, the first step is to smooth the problem into a solvable form. Then, the smoothed problem can be rewritten as a penalty function which can be solved using standard optimization algorithms.

–Step 1: General Form

We start by defining a Linear Bilevel Programming problem. It will take on the form of an optimization function with constraints. One of these constraints will be another optimization function with one or more constraints.

Let $x \in X \subset \mathbb{R}^n, y \in Y \subset \mathbb{R}^m, F : X \times Y \to \mathbb{R}^1, f : X \times Y \to \mathbb{R}^1$

The General Form of a linear BLP can be written as:

$$\begin{aligned} &\min_{x \in X} F(x,y) = c_1 x + d_1 y \\ &\text{s.t. } A_1 x + B_1 y \leq b_1 \\ &\qquad \min_{y \in Y} f(x,y) = c_2 x + d_2 y \\ &\qquad \text{s.t. } A_2 x + B_2 y \leq b_2 \\ &c_1, c_2 \in \mathbb{R}^n, d_1, d_2 \in \mathbb{R}^m, b_1 \in \mathbb{R}^p, b_2 \in \mathbb{R}^q, \\ &A_1 \in \mathbb{R}^{p \times n}, B_1 \in \mathbb{R}^{p \times m}, A_2 \in \mathbb{R}^{q \times n}, B_2 \in \mathbb{R}^{q \times m} \end{aligned} \tag{1}$$

–Step 2: KKT Conditions

According to the above introduction, we can use the Karush-Kuhn-Tucker (KKT) conditions to replace the lower level linear programming problem with its KKT optimaility conditions and get the following one-level problem.

$$\begin{aligned} &\min\ c_1 x + d_1 y, \\ &\text{s.t. } A_1 x + B_1 y \leq b_1, \\ &\qquad A_2 x + B_2 y \leq b_2, \\ &\qquad u B_2 - v = -d_2, \\ &\qquad u(b_2 - A_2 x - B_2 y) + vy = 0, \\ &x \geq 0, y \geq 0, u \geq 0, v \geq 0. \end{aligned} \tag{2}$$

–Step 3: Equivalence Relation

When the problem is in this form (2), it is considered a mathematical program with complementary constraint. The regulatory assumptions necessary for handling smooth optimization programs are never satisfied, so we cannot do much with it in its current form. With the constraints in this form, we can solve the problem; however, these equalities are non-smooth. The minimum value of these 2 variables will always be 0; these discrete variables are not effective to solve using optimization algorithms. We will use a smoothing method capable of smoothing out these equalities. However, first we must reformulate problem (2) to the following non-smooth equivalent reformulation:

$$\begin{aligned} &\min\ c_1 x + d_1 y, \\ &\text{s.t. } A_1 x + B_1 y \leq b_1, \\ &\qquad A_2 x + B_2 y \leq b_2, , \\ &\qquad u B_2 - v = -d_2, \\ &\qquad -2min(u, b_2 - A_2 x - B_2 y) = 0, \\ &\qquad -2min(v, y) = 0, \\ &x \geq 0. \end{aligned} \tag{3}$$

The ”min” operator is applied component-wise to the vectors. Problem (3) is a non-smooth optimality problem, and it is not good to use PSO or HPS to solve it in this form. Fortunately, there exists a smoothing method for problem (3).

–Step 4: Smoothing

To successfully rewrite a linear BLP, the function must have continuous derivatives. It is clear that this function is non-smooth, since the minimum operators will always be equivalent to 0. Luckily, there does exist a function that is capable of preforming a smoothing method to solve the Linear BLP problem.

Let $\varepsilon \in \mathbb{R}$ be a parameter. Define the equation $\phi_\varepsilon : \mathbb{R}^2 \to \mathbb{R}$ by

$$\phi_\varepsilon(a,b) = \sqrt{(a-b)^2 + 4\varepsilon^2} - (a+b) \tag{4}$$

Proposition 1.*For every $\varepsilon \in \mathbb{R}$ we have $\phi_\varepsilon(a,b) = 0 \Leftrightarrow a \geq 0, b \geq 0, ab = \varepsilon^2$*

In general, the purpose of ε is to allow for a problem to always be differentiable. If ε was not in the method, and both a and b were equal to zero, then taking the derivative would lead to an error. Therefore, ε is usually set to a low value. So, if $\varepsilon \neq 0, \phi_\varepsilon(a,b)$ is always differentiable. However, even if ε=0, then $\phi_\varepsilon(a,b) = 0 \Leftrightarrow a \geq 0, b \geq 0, ab = 0$. So, for every (a,b), $\lim_{\varepsilon \to 0} \phi_\varepsilon(a,b) = -2\min(a,b)$. Therefore, $\phi_\varepsilon(a,b)$ is a smooth perturbation of the complementary conditions. Then, problem (3) can be approximated by:

$$\begin{aligned} &\min\ c_1 x + d_1 y, \\ &\text{s.t. } A_1 x + B_1 y \leq b_1, \\ &\quad u B_2 - v = -d_2, \\ &\quad \sqrt{[u_i - (b_2 - A_2 x - B_2 y)_i]^2 + 4\varepsilon^2} \\ &-u_i - (b_2 - A_2 x - B_2 y)_i = 0, \qquad i = 1, \ldots, q, \\ &\quad \sqrt{(v_j - y_j)^2 + 4\varepsilon^2} - v_j - y_j = 0, \ \ j = 1, \ldots, m, \\ &x \geq 0. \end{aligned} \tag{5}$$

Using the smoothing method, we can overcome the discontinuous derivatives introduced by the KKT optimality conditions, and successfully handle smooth optimization problems. To simplify the above problem, we can write linear BLP equivalently as follows:

$$G(x,y,u,v) = \begin{pmatrix} A_1x + B_1y - b_1 \\ -x \end{pmatrix},$$

$$H(x,y,u,v) = \begin{pmatrix} uB_2 - v + d_2 \\ \phi_\varepsilon(u_i, (b_2 - A_2x - B_2y)_i), i = 1,...,q \\ \phi_\varepsilon(v_j, y_j), j = 1,...,m \end{pmatrix}$$

2.2 Penalty Method

Once the problem has been rewritten as a one-level constrained problem, it cannot be solved by a standard optimization algorithm. The majority of these algorithms require a distinct unconstrained function to run effectively. In its current form, there's no way to effectively evaluate it. So, we implement a penalty function to penalize the violations on the equalities and inequalities at point x', and rewrite the constrained problem as an unconstrained one. To calculate the fitness of the penalty function:

1.Calculate the fitness of the objective function

$$v_1(x') = F(x')$$

2.Calculate the violation of the equality constraints:

$$v_2(x') = \sum_{l=1}^{d} (H_l(x'))^2$$

3.Calculate the violation of the inequality constraints:

$$v_3(x') = \sum_{k=1}^{c} (I(G_k(x')))^2$$

where the function $I(x)$ is defined as follows:

$$I(x) = \begin{cases} 0, & x \geq 0, \\ x, & x < 0. \end{cases}$$

4.Compute the fitness value $v(x)$ using the formula:

$$v(x') = v_1(x') + c * v_2(x') + c * v_3(x')$$

where c > 0.
Problems in this form will be nonlinear and unconstrained. Which is ideal for standard optimization methods to solve.

3 Algorithms

3.1 Memetic Particle Swarm Optimization

3.1.1 Introduction

Particle Swarm Optimization (PSO) is a stochastic, population-based search algorithm that gained a lot of attention since its development in 1995. Its popularity can be attributed to its easy implementation and ability to efficiently solve a plethora of problems in science and engineering.

PSO was inspired by the analysis of social dynamics and interactions among members of organized colonies; therefore, it is categorized as a swarm based algorithm. PSO has many common key characteristics with Evolutionary Algorithms (EAs), such as Genetic Algorithms, Evolution Strategies, and Differential Evolution, so it shares many aspects of their behaviour. EA's have proved to be effective in many applications. However, there is a well-known problem regarding their local search abilities in optimization problems. More specifically, although most EAs are capable of detecting the region of attraction of the global optimum with high accuracy, unless specific procedures are incorporated in their operators. Some versions of PSO also exhibit this deficiency.

The aforementioned drawback of EAs triggered the development of Memetic Algorithms (MAs), which incorporate local search components. MAs constitute a class of metaheuristics that combines population-based optimization algorithms with local search procedures. More specifically, MAs consist of a global component, which is responsible for a rough search of the search space and the detection of the most promising regions. MA's also have a local search component, which is used for probing the detected promising regions, in order to obtain solutions with high accuracy. EAs have been used as the global component in MAs with Simulated Annealing and random local search. MAs have proved to be an unrivalled methodology in several problem domains.

We will be using an algorithm that combines PSO with local search methods, resulting in an efficient Memetic PSO scheme. The section is organized as follows: The proposed approach will be displayed in Section 3.1.2 and Section 3.1.3 is dedicated to the purpose of using MPSO to solve linear BLP.

3.1.2 Definition of MPSO

The Algorithm consists of the following steps:

Algorithm 1 MPSO scheme

0. **Input**: N(Population Size), χ, c_1, c_2,a,b (lower and upper bounds), F (objective function).
1. **Set** $t = 0$.
2. **Initialize** $x_i^{(t)}, v_i^{(t)} \in [a,b], p_i^{(t)} \leftarrow x_i^{(t)}$, for $i = 1,\ldots,N$.
3. **Evaluate** $F(x_i^{(t)})$
4. **Determine** the indicies g_i, for $i = 1,\ldots,N$.
5. **While** (stopping criterion is not satisfied) **do**
   (a) **Update** the velocities $v_i^{(t+1)}$, for $i = 1,\ldots,N$.
   (b) **Set** $x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)}$, for $i = 1,\ldots,N$.
   (c) **Constrain** each particle x_i in $[a,b]$.
   (d) **Evaluate** $F(x_i^{(t+1)})$, for $i = 1,\ldots,N$.
   (e) **If** $F(x_i^{(t+1)}) < F(p_i^{(t)})$ **Then** $p_i^{(t+1)} \leftarrow x_i^{(t+1)}$.
   (f) **Else** $p_i^{(t+1)} \leftarrow p_i^{(t)}$
   (g) **Update** the indicies g_i.
   (h) **When** (local search is applied) **Do**
      i. **Choose** (according to one of the Schemata 1-3) $p_q^{(t+1)}, q \in 1,\ldots,N$.
      ii. **Apply** local search on $p_q^{(t+1)}$ and obtain a new solution, y.
      iii. **If** $F(y) < F(p_q^{(t+1)})$ **Then** $p_q^{(t+1)} \leftarrow y$.
   (i) **End When**
   (j) **Set** $t = t+1$.
6. **End While**

–Step 1: Determine user-defined variables
The MPSO algorithm requires many different variables to function. Some of these variables are problem dependent, such as the lower and upper bounds of the components (a and b, respectively) as well as the objective function being evaluated. However, some variables must be set by the user. The χ, c_1, c_2 variables are needed to calculate the velocities of the particles (Step 5). We set the variables to $0.79, 2.05$, and 2.05, respectively.

–Step 2: Initialize the Population
The initial positions and velocities of the particles are then generated. The positions x of the particles are generated first. Each of the n-dimensional components of the particle are randomly generated within the feasible region $[a,b]$. So, every component is random and within a set range. This process is repeated N times; once for every particle in the swarm (population). This event is repeated for the initial velocities v of the particles, such that there are N sets of velocities where every n-dimensional component is within the range of $[a,b]$. We then save the positions of the particles in p, which stores each particles best known position.

–Step 3: Evaluate the Fitness
Once the initial positions and velocities of the particles are generated, we must evaluate the fitness of each N particle using function F. The constrained linear bilevel programming problem has been transformed into a unconstrained linear problem. Therefore, it is possible to use MPSO to solve for the fitness of each particle.

–Step 4: Determine the Indices
The indices refer to the position of the best particle relative to the neighbourhood of each individual particle of the swarm. These best positions are used to calculate the velocities of particles in Step 5. There are two different types of indices that are used in this algorithm.
The first method to calculate the indices is the easiest. It is simply the position of the most fit particle in the swarm. This is the basis for the PSOg algorithm, where the indices is the global minimum.
The second method to calculate the indices is by finding the position of the particle with the best fitness in every individual particle's neighbourhood. To start, please note that a particle's neighbourhood is based on its index in the population and not based on the particle's physical position. To calculate this, we start by taking the neighbourhood radius, and selecting all the particles that are within that radius, which is based off a ring topology. For example, if we have a radius of 2, and we want to find the neighbourhood of particle 13 (the 13^{th} particle generated in Step 2), the particle's neighbourhood consists of particles 11, 12, 13, 14, and 15. The value of each index is the position of the particle with the best fitness within a given neighbourhood. This is the basis for the PSOl algorithm, where the indices calculate a local minimum.

–Step 5: Update the Velocities
The velocities of the particles is the most important part of this algorithm as it defines how the particles will move. Velocity is calculated by:

$$v_i^{(t+1)} = \chi[v_i^{(t)} + c_1 r_1 (p_i^{(t)} - x_i^{(t)}) + c_2 r_2 (p_{gi}^{(t)} - x_i^{(t)})]$$

where $i = 1,2,\ldots,N$. This equation states that the velocity is essentially calculated from the addition of the old velocity, the difference between a particle's best and current position, and the difference between the current particle and the best particle in the current particle's neighbourhood. The variables χ, c_1, c_2 are user defined variables. r_1, r_2, are random variables in the range $(0,1)$. This process is done until all N particles have updated their velocities. These variables affect how the MPSO algorithm will function.

–Step 6: Calculate new positions
Once we have the velocities of the particles we can find the new positions. To get the new position we simply add the current position of a particle with the new velocity of the particle. This step is done on a component-wise basis. This process is repeated N times, so every particle's components are updated.

–Step 7: Evaluate the fitness
Now that the positions of the particles have been updated, we must calculate the new fitness of each particle. Similarly to step 3, each N particle is run

through the objective function and the fitness value for each particle is recorded.

–**Step 8: Updating Variables**

Firstly, we need to record the best positions of each particle. We compare the current fitness with the best fitness of each particle. If the new fitness is better, we save the positions of the new, better particle.

Secondly, we must once again update the indices in the manner described in Step 4.

–**Step 9: Local Search**

At this step, we apply a local search to a select amount of particles to improve their fitness by making slight adjustments to their positions. There are 3 different methods to allow a particle to be affected by a local search. The local search we used in this algorithm was the Random Walk with Directional Exploitation.

There are 3 methods in which a particle is chosen to undergo the local search. The first method is to accept only the particle with the best fitness. This method assumes the particle with the best fitness is the particle that is closest to the optimum solution, so that particle undergoes a local search. The second method is that every particle has a small chance to have a local search preformed on it. This allows all particles to have a chance to undergo a small improvement. The final method is a combination of both of the other methods. Every particle has a chance to undergo a local search, but the particle with the best fitness will automatically go through it. Which local search schemata is chosen is problem dependent.

Random Walk with Directional Exploitation is the local search technique we used for this algorithm. It follows the general formula:

$$x^{(t+1)} = x^{(t)} + \lambda * z^{(t)}$$

Let $x^{(t)}$ be the approximation of the optimal solution at the tth iteration. Then, the new value (approximation), $x^{(t+1)}$ at the (t+1)th iteration will be calculated where λ is a prescribed step-length, and $z^{(t)}$ is a unit-length random vector. To calculate the approximation, we follow the algorithm:

Algorithm 2 Random Walk with Directional Exploitation

1: **Initialize** $t = 0$, initial point $x^{(1)}$, and $\lambda = \lambda_{init}$.
2: **Compute** $F = f(x^{(1)})$, where f is the objective function.
3: **While** $t \leq t_{max}$
4: (a) **Set** $t = t + 1$
5: (b) **Generate** a unit-length random vector $z^{(t)}$
6: (c) **Compute** $F' = f(x^{(t)} + \lambda * z^{(t)})$
7: (d) **If** $F' < F$ **then** set $x^{(t+1)} = x^{(t)} + \lambda * z^{(t)}, t = t + 1, \lambda = \lambda_{init}$, and $F = F'$ and goto Step 3(c).
8: (e) **Else If** $F' > F$ **then** set $x^{(t+1)} = x^{(t)}$, reduce $\lambda = \lambda/2$, and goto Step 3(a).
9: (f) **Else** $F' = F$ **then** set $x^{(t+1)} = x^{(t)}$, and goto Step 3(a).
10: **End While**

We start with the initial point $x^{(1)}$ which is the position of the particle chosen to have the local search applied to it. We then obtain the fitness of that point. We then generate a random unit-length vector multiplied by a scalar value to represent a suitable range for the search. If $x^{(t)} + \lambda * z^{(t)}$ has a better fitness value than the current particle position, then the new approximation and fitness will replace the previous iteration, the scalar value is reset, and the unit-length vector is preserved. If the approximation's fitness value is worse that the current approximation, the scalar value is reduced by half and a new unit-length vector is generated. If the approximation's fitness value is equal to the current approximation, then a new unit-length vector is generated. This process is repeated until a certain number of iterations has occurred.

–**Step 10: Stopping Criteria**

Once the local search has been completed, the iteration counter t is then incremented for the next iteration of the algorithm. This is where the stopping criteria for the algorithm is evaluated. If the criteria is not met the algorithm jumps to step 5, and the next iteration of the algorithm begins. If the criteria is met, then the algorithm returns the position of the particle with the best fitness.

3.1.3 MPSO

MPSO is an algorithm that combines the effectiveness of the PSO algorithm with the precision of a local search. MPSO was chosen for this paper because of its ability to effectively handle problems with multiple components well. It is also capable of accurate results in decimal-value components.

There are multiple advantages to using MPSO. As the velocity is calculated component-wise, it has a higher chance of being successful of finding the optimal solution for a problem that has a high number of components. There are few variables to manipulate; we set out constants to the default values, $\chi = 0.79, c_1, c_2 = 2.05$ [17]. It is a straight forward algorithm that is easy to implement. Finally, it is general, so it can be used in multiple different fields for multiple purposes.

On the other hand, MPSO has disadvantages as well. The method easily suffers from the partial optimism, which causes the less exact of the regulation of its velocity and direction. The algorithm cannot work out the problems of scattering.

3.2 Heuristic Pattern Search

3.2.1 Introduction

Pattern Search (PS) is a nonlinear search algorithm created by Hooke and Jeeves [8] in 1961. The Pattern

Search Algorithm has been widely used in a nonlinear programming context, emerging as an efficient algorithm for solving unconstrained, bound constrained, as well as linearly or nonlinearly nonsmooth constrained problems.

The Pattern Search has two main points. The first is a *Exploratory Move* at a base point, and the second is a *Pattern Move* around a successful point after an Exploratory move.

We will be using a Heuristic Pattern Search (HPS) algorithm designed by Isabel A. C. P. Espirito Santo and Edite M. G. P. Fernandes [20] in 2011. It combines the usual pattern and exploratory moves of the Hooke and Jeeves method with a random approximate descent search. In this way, no information of the evaluated function's derivatives are required to generate a descent move.

The algorithm for HPS will be defined in Section 3.2.2 is dedicated to the purpose of using MPSO to solve linear BLP.

3.2.2 Definition of HPS

The Algorithm consists of the following steps:

Algorithm 3 The main steps of Heuristic Pattern Search Algorithm

0. **Input**: $x^{(1)}, \gamma_\Delta, a, b$ (lower and upper bounds), F (objective function).
1. **Set** $k = 1$.
2. **Evaluate** $F(x^{(1)})$ and set $F(x^{(0)}) = F(x^{(1)})$
3. **Generate** initial step vector Δ_0 from γ_Δ
4. **While** (stopping criterion is not satisfied) **do**
   if $F(x^{(k-1)}) > F(x^{(k)})$ then
      (a) **Preform** pattern move to get point p_k.
      (b) **Generate** a random descent move d_k at point p_k.
      (c) **Set** $x^{(k+1)} = p^k + \lambda d_k$
      (d) **Constrain** $x^{(k+1)}$ in the feasible region $[a, b]$
   end if
   if $F(x^{(k-1)}) \leq F(x^{(k)})$ then
      (e) **Preform** an exploratory move to get point $s_{(k)}$
      (f) **Set** $x^{(k+1)} = x^{(k)} + s_k$
      (g) **Constrain** $x^{(k+1)}$ in the feasible region $[a, b]$
   end if
   Set $k = k + 1$.
5. **End While**

–Step 1: Determine user-defined variables
The HPS algorithm doesn't require many user defined variables to enter. Some of these variables are problem dependent, such as the lower and upper bounds of the components (a and b respectively), as well as the objective function F being evaluated. The γ_Δ variable is used to determine the initial step size of the Δ_0 variable which will be defined greater in step 3. The initial point $x^{(1)}$ must also be defined.

–Step 2: Evaluate the Objective function at the initial point
The objective function is evaluated at the user-generated initial point $x^{(1)}$

–Step 3: Generate initial Step Length Δ_0
The step length is necessary to preform the exploratory move. This will be discussed greater in Step 7. To calculate each component of Δ_0, we preform the following calculation. **If** $x_i^{(1)} \neq 0$ then $(\Delta_0)_i = \gamma_\Delta * x_i^{(1)}$; otherwise $(\Delta_0)_i = \gamma_\Delta$ for $i = 1, 2, \ldots, N$ where $\gamma_\Delta > 0$.

–Step 4: Preform a pattern move
When the current iterate (i.e. $x^{(k)}$) is an improvement over the previous one, we assume that if we made the same move again, it would improve the value again. In other words, the pattern move, $p_k = x^{(k)} + (x^{(k)} - x^{(k-1)})$.

–Step 5: Generate a random descent direction
Here, we describe a strategy to generate an approximate descent direction, d_k, for the objective function F, at the point p_k. This is important since experience shows that search directions that are parallel to the coordinate axes may be uphill at points of the search region. We randomly generate two points y_1 and y_2 in the neighbourhood of p_k, such a way that $||p_k - y_i||$ where $i = 1, 2$ for a sufficiently small positive value of ε, a vector with a high probability of being a descent direction for the objective function at p_k is generated by:

$$d_k = -\frac{1}{\sum_{j=1}^{2} |\Delta f_j|} \sum_{i=1}^{2} (\Delta f_i) \frac{p_k - y_i}{||p_k - y_i||}$$

where $\Delta f_j = f(p_k) - f(y_j)$. We set the value ε to be 0.001.

–Step 6: Calculate x^{k+1} at point p_k
We now set $x^{(k+1)} = p_k + \lambda d_k$ where $\lambda \in (0,1]$. We initially set the value of λ to be 1. If the objective function at this point is not superior to the objective value at the previous iterate x^k, then the value of λ is halved. This process is repeated up to 5 times. If a superior value is not found, then we preform an exploratory move.

–Step 7: Preform an exploratory move
We now preform an exploratory move at the point $x^{(k)}$. Firstly, we set the value of $x^{(k+1)} = x^{(k)}$. For each component, we calculate two variables with the values $(x^{(k+1)})_i \pm (\Delta)_i$ where delta is the step length vector. If either of these two variables has an objective value better than $x^{(k+1)}$, to the best value. If, after all N components, the objective value has not changed (i.e. $x^{(k+1)} = x^{(k)}$) then we reduce each component of Δ by half to increase our odds of an effective exploratory move in later iterations.

–Step 8: Stopping Criteria

If the current iterate $x^{(k)}$ is within the error goal of 10^{-4} of the optimal value of F, then the algorithm terminates. If the algorithm preforms 5000 iterations before the optimal value is reached, then we assume the process failed and terminate the algorithm.

3.3 Proposed Algorithm

3.3.1 Introduction

We introduce a compound search algorithm that combines the Memetic Particle Search Algorithm and the Heuristic Pattern Search algorithm to solve Linear Bilevel Programming Problem. The algorithm will use the foundation of the MPSO Algorithm, but use the HPS as the local search instead of the Random Walk with Directional Exploit as described in Section 3.1.2. We will be using the same schemata as Random Walk algorithm to determine which particles will undergo the local search in each problem. We will use the same default parameters for each HPS as described in Section 3.2.2. We hope that this new algorithm will outpreform both the PSO and RWMPSO in both the number or successful solutions and mean number of iterations to solve the problem.

3.3.2 Definition of MPSOwPS

The Algorithm consists of the following steps:

Algorithm 4 MPSOwPS scheme

0. **Input**: N(Population Size), χ, c_1, c_2,a,b (lower and upper bounds), F (objective function).
1. **Set** $t = 0$.
2. **Initialize** $x_i^{(t)}, v_i^{(t)} \in [a,b], p_i^{(t)} \leftarrow x_i^{(t)}$, for $i = 1,\ldots,N$.
3. **Evaluate** $F(x_i^{(t)})$
4. **Determine** the indicies g_i, for $i = 1,\ldots,N$.
5. **While** (stopping criterion is not satisfied) **do**
   (a) **Update** the velocities $v_i^{(t+1)}$, for $i = 1,\ldots,N$.
   (b) **Set** $x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)}$, for $i = 1,\ldots,N$.
   (c) **Constrain** each particle x_i in $[a,b]$.
   (d) **Evaluate** $F(x_i^{(t+1)})$, for $i = 1,\ldots,N$.
   (e) **If** $F(x_i^{(t+1)}) < F(p_i^{(t)})$ **Then** $p_i^{(t+1)} \leftarrow x_i^{(t+1)}$.
   (f) **Else** $p_i^{(t+1)} \leftarrow p_i^{(t)}$
   (g) **Update** the indicies g_i.
   (h) **When** (local search is applied) **Do**
      i. **Choose** (according to one of the Schemata 1-3) $p_q^{(t+1)}, q \in 1,\ldots,N$.
      ii. **Apply** Heuristic Pattern Search on $p_q^{(t+1)}$ and obtain a new solution, y.
      iii. **If** $F(y) < F(p_q^{(t+1)})$ **Then** $p_q^{(t+1)} \leftarrow y$.
   (i) **End When**
   (j) **Set** $t = t+1$.
6. **End While**

4 Numerical Results

This is an attempt to experimentally show that the defined MPSOwPS scheme can outperform the standard PSO and MPSO algorithms. To achieve this, we have considered 6 BLP problems and we have experimentally considered the parameters used by the proposed approach. For all test problems, the PSO parameters were set to their default values $\chi = 0.79, c_1, c_2 = 2.05$ [3]. The remaining parameters, such as the number of iterations and the step length of the local search method used, were problem dependent and, thus, individually specified for each test problem. The RWMPSO algorithm uses the Random Walk with Direction Exploitation as the local search variant, which is described in Section 3.1.2. The MPSOwPS algorithm uses the Heuristic Pattern Search described in Section 3.2.2 as the local Search variant. In both cases, the local search variant is used as described in step 9 of the MPSO algorithm.

Table 1: Parameters for the test problems

Problem	Dimension	Range	Error Goal
Ex. 1	2	$(0,10)^2$	10^{-4}
Ex. 2	3	$(0,10)^3$	10^{-4}
Ex. 3	2	$(0,20)^2$	10^{-4}
Ex. 4	2	$(0,20)^2$	10^{-4}
Ex. 5	2	$(0,10)^2$	10^{-4}
Ex. 6	5	$(0,10)^5$	10^{-4}

4.1 Examples

The problems we used are:
Test problem 1 [11].

$$\begin{aligned} &\min_{x\geq 0} F(x,y) = x - 4y \\ &\text{s.t. } \min_{y\geq 0} f(x,y) = y \\ &\quad \text{s.t. } -x - y < -3, \\ &\qquad -2x + y \leq 0, \\ &\qquad 2x + y \leq 12, \\ &\qquad 3x - 2y \leq 4, \\ &x, y \geq 0. \end{aligned}$$

Test problem 2 [11].

$$\begin{aligned} &\min_{x\geq 0} F(x,y) = 4x_1 + y_1 + y_2 \\ &\text{s.t. } \min_{y\geq 0} f(x,y) = x + 3y_1 \\ &\quad \text{s.t. } x + y_1 + y_2 \leq \tfrac{25}{9}, \\ &\qquad x + y_1 \leq 2, \\ &\qquad y_1 + y_2 \leq \tfrac{8}{9}, \\ &x, y \geq 0. \end{aligned}$$

Test problem 3 [23].

Table 2: Parameter settings of RWMPSOg for the test problems

Problem	Pop. Size	Iter	Step	Best	Prob	Freq
Ex. 1	25	5	1.0	yes	-	1
	50	5	1.0	yes	-	1
Ex. 2	25	10	1.0	-	0.1	1
	50	10	1.0	-	0.1	1
Ex. 3	25	5	2.0	-	0.1	1
	50	5	3.0	-	0.1	1
Ex. 4	25	10	1.0	yes	-	1
	50	5	1.0	yes	-	1
Ex. 5	25	5	3.0	-	0.1	1
	50	5	3.0	-	0.1	1
Ex. 6	25	5	1.0	-	0.1	1
	50	5	1.0	-	0.1	1

Table 3: Parameter settings of RWMPSOl for the test problems

Problem	Pop. Size	Iter	Step	Best	Prob	Freq
Ex. 1	25	10	1.0	yes	-	1
	50	10	1.0	yes	-	1
Ex. 2	25	10	1.0	yes	-	1
	50	10	1.0	yes	-	1
Ex. 3	25	5	1.0	yes	-	1
	50	5	1.0	yes	-	1
Ex. 4	25	10	1.0	yes	-	1
	50	10	1.0	yes	-	1
Ex. 5	25	5	1.0	yes	-	1
	50	5	1.0	yes	-	1
Ex. 6	25	5	1.0	yes	-	1
	50	5	1.0	yes	-	1

Table 4: Parameter settings of PSOgPS for the test problems

Problem	Pop. Size	Iter	Δ_γ	Best	Prob	Freq
Ex. 1	25	5	1	-	0.1	1
	50	5	1	-	0.1	1
Ex. 2	25	5	1	-	0.1	1
	50	5	1	-	0.1	1
Ex. 3	25	10	1	-	0.1	1
	50	10	1	-	0.1	1
Ex. 4	25	5	0.01	-	0.1	5
	50	5	0.01	-	0.1	5
Ex. 5	25	5	1	-	0.1	5
	50	5	1	-	0.1	5
Ex. 6	25	5	100	-	0.2	1
	50	5	100	-	0.2	1

$$\begin{aligned}
&\min_{x\geq 0} F(x,y) = x+3y\\
&\text{s.t. } \min_{y\geq 0} f(x,y) = x-3y\\
&\quad \text{s.t. } -x-2y \leq -10,\\
&\qquad x-2y \leq 6,\\
&\qquad 2x-y \leq 21,\\
&\qquad x+2y \leq 38,\\
&\qquad -x+2y \leq 18,\\
&sx,y \geq 0.
\end{aligned}$$

Table 5: Parameter settings of PSOlPS for the test problems

Problem	Pop. Size	Iter	Δ_γ	Best	Prob	Freq
Ex. 1	25	5	0.01	-	0.1	5
	50	5	0.01	-	0.1	5
Ex. 2	25	5	0.01	-	0.1	1
	50	5	0.01	-	0.1	1
Ex. 3	25	10	1	-	0.1	1
	50	5	1	-	0.1	1
Ex. 4	25	5	1	yes	-	1
	50	5	1	yes	-	1
Ex. 5	25	10	1	-	0.1	1
	50	10	1	-	0.1	1
Ex. 6	25	10	100	yes	-	5
	50	10	100	yes	-	5

Test problem 4 [9].

$$\begin{aligned}
&\min_{x\geq 0} F(x,y) = -2x+11y\\
&\text{s.t. } \min_{y\geq 0} f(x,y) = -x-3y\\
&\quad \text{s.t. } x-2y \leq 4,\\
&\qquad 2x-y \leq 24,\\
&\qquad 3x+4y \leq 96,\\
&\qquad x+7y \leq 126,\\
&\qquad -4x+5y \leq 65,\\
&\qquad -x-4y \leq -8,\\
&x,y \geq 0.
\end{aligned}$$

Test problem 5 [12]

$$\begin{aligned}
&\min_{x\geq 0} F(x,y) = 2x-y\\
&\text{s.t. } \min_{y\geq 0} f(x,y) = x+2y\\
&\quad \text{s.t. } 3x-5y \leq 15,\\
&\qquad 3x-y \leq 21,\\
&\qquad 3x+y \leq 27,\\
&\qquad 3x+4y \leq 45;\\
&\qquad x+3y \leq 30,\\
&x,y \geq 0.
\end{aligned}$$

Example 6 [12].

$$\begin{aligned}
&\min_{x\geq 0} F(x,y) = -8x_1-4x_2+4y_1-40y_2-4y_3\\
&\text{s.t. } \min_{y\geq 0} f(x,y) = x_1+2x_2+y_1+y_2+2y_3\\
&\quad \text{s.t. } -y_1+y_2+y_3 \leq 1,\\
&\qquad 2x_1-y_1+2y_2-0.5y_3 \leq 1,\\
&\qquad 2x_1+2y_1-y_2-0.5y_3 \leq 1,\\
&x_1,x_2,y_1,y_2,y_3 \geq 0.
\end{aligned}$$

All Lagrange multipliers for each test problem generated from the KKT conditions will be within the range (0,10).

The dimension of each test problem, the range in which the particles were constrained, as well as the error goals are reported in Table 1. The maximum number of

Table 6: Results for the test problems

Example	Optimal Solution	Pop. Size	Algorithm	min	mean	max	std	suc.
Ex. 1	**-12.0**	25	PSOg	21956	21956.00	21956	0.00	1
			RWMPSOg	2630	11909.20	32319	11321.50	10
			MPSOgPS	484091	862654.16	1911143	314527.8291	25
			PSOl	84562	84562.00	84562	0.00	1
			RWMPSOl	5149	52444.65	102163	28423.35	17
			MPSOlPS	39898	188629.7838	294707	59871.2279	37
		50	PSOg	19269	23997.50	28726	6687.11	2
			RWMPSOg	4408	19518.00	45652	16527.00	12
			MPSOgPS	1131852	1767604.038	2839058	428948.6053	26
			PSOl	9528	116753.75	230628	122920.13	4
			RWMPSOl	6378	99420.17	213941	58960.10	30
			MPSOlPS	22136	370705.1739	589918	127135.637	46
Ex. 2	**76/9**	25	PSOg	0	0.00	0	0.00	0
			RWMPSOg	14527	17451.00	25938	3352.00	11
			MPSOgPS	212711	324282.4667	549929	116461.751	15
			PSOl	8352	24980.75	36224	12073.97	4
			RWMPSOl	4179	22010.89	64263	14548.35	28
			MPSOlPS	57415	319293.0435	1095995	170656.2658	46
		50	PSOg	35140	37083.00	39026	2747.82	2
			RWMPSOg	11222	39370.00	168679	43285.00	12
			MPSOgPS	598262	1034729.208	2373310	386682.8849	24
			PSOl	16672	51549.50	99630	30026.61	8
			RWMPSOl	11220	49009.19	206057	38181.89	43
			MPSOlPS	67815	433518.04	1275799	280286.2435	50
Ex. 3	**49.0**	25	PSOg	10411	18725.00	27039	11757.77	2
			RWMPSOg	3364	14280.33	32753	9172.27	12
			MPSOgPS	206993	654382.0851	2263118	280894.9052	47
			PSOl	1991	50532.00	105417	41147.95	5
			RWMPSOl	3621	44443.00	115894	32310.42	11
			MPSOlPS	73763	712310.7111	2228091	391066.3452	45
		50	PSOg	12143	30582.75	76275	30709.95	4
			RWMPSOg	4976	24703.58	37391	9051.75	13
			MPSOgPS	1214940	1520398.4	3094784	303367.5573	50
			PSOl	96253	159673.19	235699	38148.71	16
			RWMPSOl	5456	101956.33	238786	75974.58	24
			MPSOlPS	107333	796422.26	2301052	354935.9097	50

Table 7: Results for the test problems

Example	Optimal Solution	Pop. Size	Algorithm	min	mean	max	std	suc.
Ex. 4	**85.0855**	25	PSOg	0	0.00	0	0.00	0
			RWMPSOg	5181	13026.00	21150	4959.42	9
			MPSOgPS	76966	104733	127259	19162.88678	8
			PSOl	36091	50692.67	68600	16504.68	3
			RWMPSOl	7118	34177.20	64309	15886.63	10
			MPSOlPS	30639	135748.6	358613	102406.845	11
		50	PSOg	0	0.00	0	0.00	0
			RWMPSOg	8019	17759.93	24507	5645.04	14
			MPSOgPS	148115	213558.25	257197	30386.3341	12
			PSOl	62983	91613.40	116631	19398.60	5
			RWMPSOl	18494	75971.56	162588	38847.80	16
			MPSOlPS	28474	176125.5	376307	103019.1394	21
Ex. 5	**9.29**	25	PSOg	2339	11207.82	35793	10845.32	17
			RWMPSOg	3424	9346.00	54406	14945.07	20
			MPSOgPS	9886	98422.93878	190839	48108.87983	49
			PSOl	4564	36763.26	116614	41452.95	19
			RWMPSOl	3503	28318.81	108453	33708.77	47
			MPSOlPS	83559	607180.6739	2118857	590863.9601	48
		50	PSOg	3791	33760.96	438839	89111.42	23
			RWMPSOg	5393	23984.83	209544	45591.71	24
			MPSOgPS	21983	187582.64	339791	81769.56104	50
			PSOl	7283	46233.67	218982	66880.12	24
			RWMPSOl	5541	38014.63	231933	59535.24	48
			MPSOlPS	122538	576143.62	3576731	715347.5022	50
Ex. 6	**-29.20**	25	PSOg	21852	27576.50	33301	8095.67	2
			RWMPSOg	20557	34809.83	51166	7928.80	12
			MPSOgPS	588770	1019468.417	2091136	373338.552	24
			PSOl	78795	102168.17	119980	15668.36	6
			RWMPSOl	21730	60448.76	89965	15262.49	38
			MPSOlPS	20262	95807.09302	193536	32869.51536	43
		50	PSOg	32875	34049.50	35410	1233.53	4
			RWMPSOg	31738	46758.41	59498	8502.48	17
			MPSOgPS	1502884	2332604.286	4343871	794042.398	21
			PSOl	103670	183889.94	247482	45830.86	17
			RWMPSOl	19881	119160.88	189722	35665.00	48
			MPSOlPS	17613	131502.2979	310069	51497.75459	48

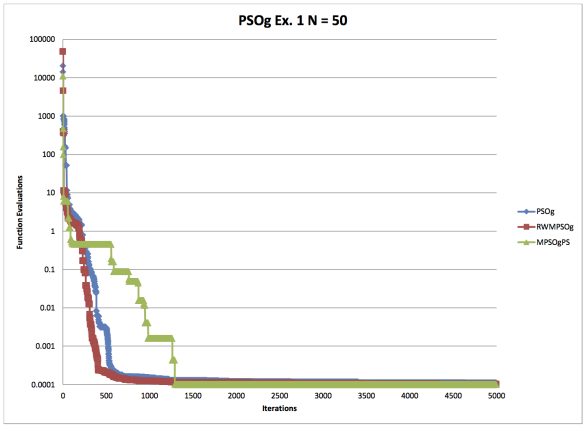
iterations for every problem was equal to 5000. For all problems, two different population sizes were considered, 25 and 50. The global and the local PSO variants (denoted as PSOg and PSOl, respectively), were equipped with RWDE, resulting in the global and local RWMPSO variants (denoted as RWMPSOg and RWMPSOl, respectively), and applied on all test problems. For each test problem, 50 independent experiments were conducted. An experiment was considered successful if the desired error goal was achieved within the maximum number of iterations.

The configuration of RWDE was problem dependent. The parameter settings of RWMPSOg and RWMPSOl for the unconstrained problems are reported in Tables 2 and 3, respectively. The first column of the tables denotes the problem, while second column stands for the swarm size. The third and fourth column report the number of iterations and initial step size used by RWDE, respectively. The fifth column has the value yes in the cases where RWDE was applied only on the best particle of the swarm. On the other hand, if RWDE was applied on the best position of each particle with a probability, then this probability is reported in column six. Finally, the last column shows the frequency of application of the local search. Thus, the value 1 corresponds to application of the local search at every iteration, while 20 corresponds to application every 20 iterations.

The results for the test problems are reported in Table 6 and 7. More specifically, the number of successes (out of 50 experiments), the minimum, mean, maximum, and standard deviation of the required function evaluations (evaluated only on the successful experiments) are reported.

4.2 Graphs

These are the behaviours of the PSO and MPSO algorithms for each test problem:



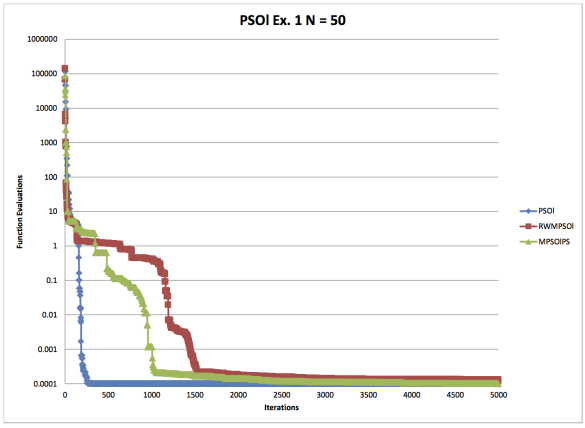


Fig. 1: Behaviour of algorithm for Ex.1

5 Conclusion

A metaheuristic approach to solving the linear Bilevel Programming Problem was proposed. The performance was investigated by several test problems. Both local and global variants of the proposed MPSOwPS scheme were tested and compared with the corresponding variants of PSO and RWMPSO. In almost all problems the MPSOwPS proved to result in a higher number of successful attempts. The direct flaw of the MPSOwPS algorithm is that it requires an ecsessive number of function evaluations compared to the RWMPSO and PSO algorithms. Overall, the MPSOwPS algorithm proved to more effective, but less efficient than either the RWMPSO and PSO algorithms for solving Linear Bilevel programming problems.

Acknowledgments

The research of the 1st author is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

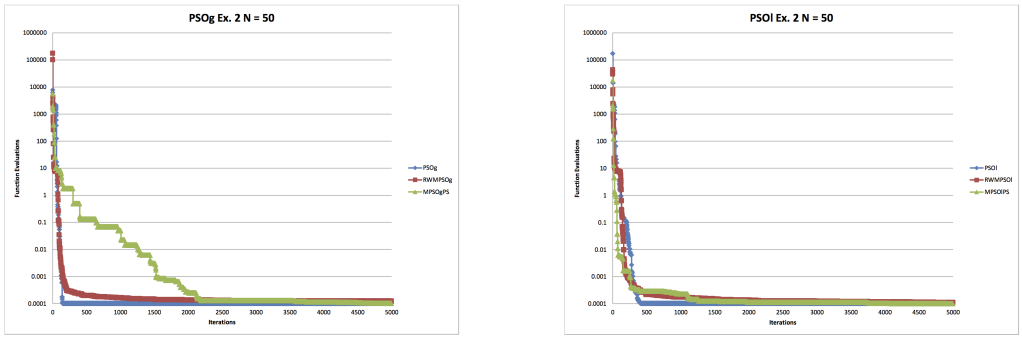


Fig. 2: Behaviour of algorithm for Ex.2

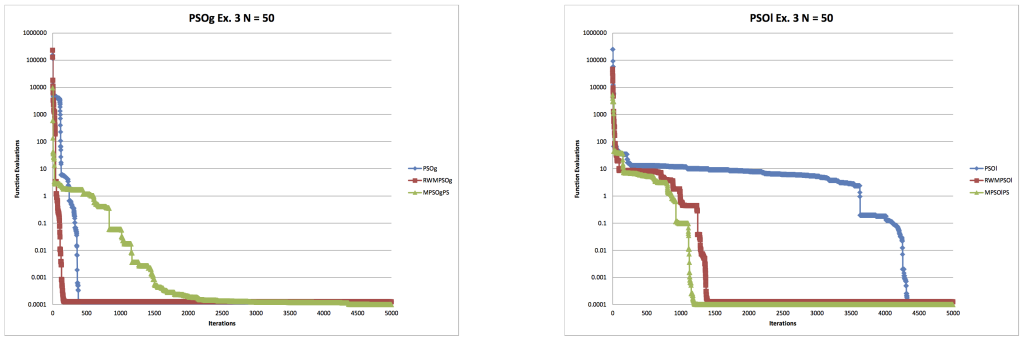


Fig. 3: Behaviour of algorithm for Ex.3

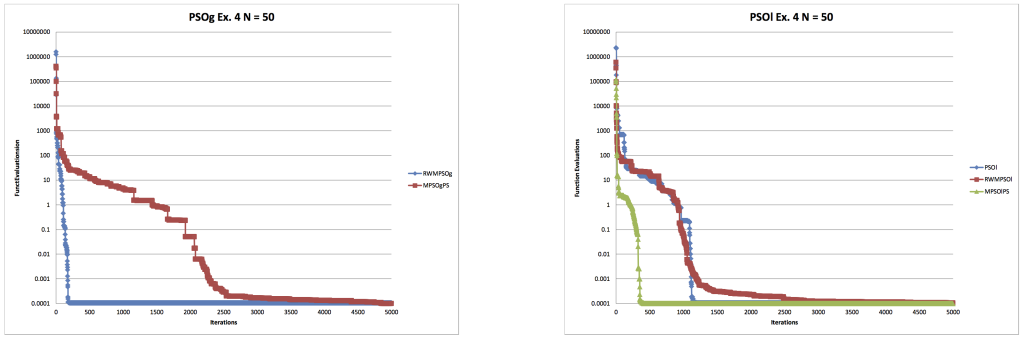


Fig. 4: Behaviour of algorithm for Ex.4

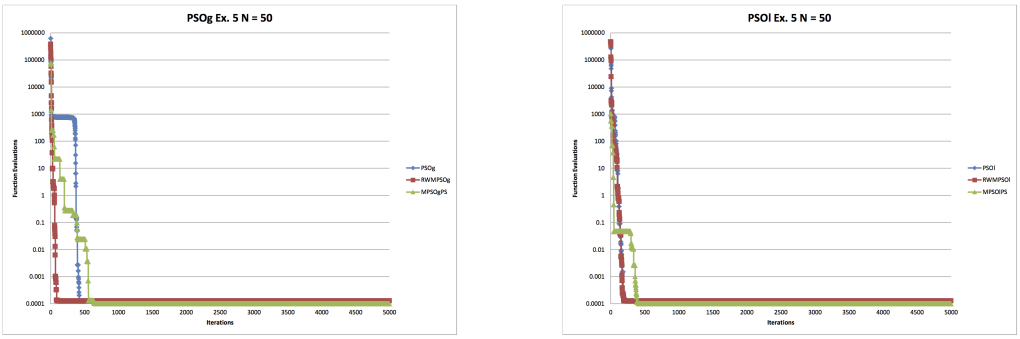


Fig. 5: Behaviour of algorithm for Ex.5

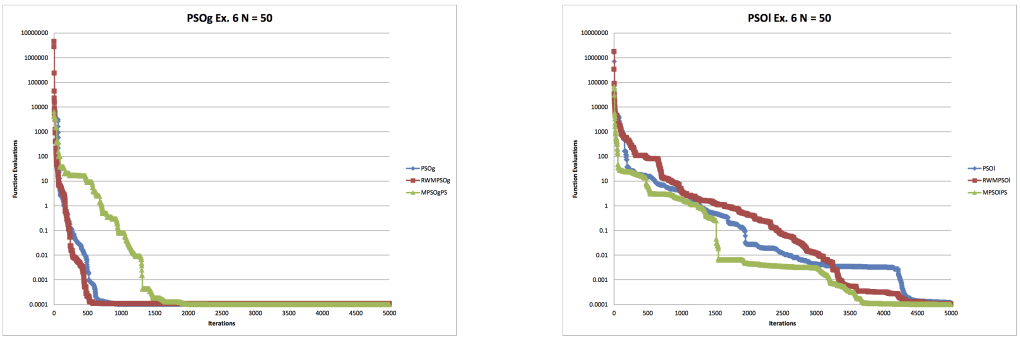


Fig. 6: Behaviour of algorithm for Ex.6

References

[1] M. A. Abido, Optimal design of power system stabilizers using particle swarm optimization, *Transactions on Energy Conversion*,**17** (2002), 406–413.

[2] D. K. Agrafiotis and W. Cedeno, Feature selection for structure-activity correlation using binary particle swarms, *Journal of Medicinal Chemistry*,**45** (2002), 1098–1107.

[3] M. Clerc and J. Kennedy, The particle swarm explosion, stability, and convergence in a multidimensional complex space., *IEEE Transactions on Evolutionary Computation*,**6** (2002), 58–73.

[4] A. R. Cockshott and B.E. Hartman, Improving the fermentation medium for Echinocandin B production. Part II: particle swarm optimization, *Process Biochemistry*,**36** (2001), 661–669.

[5] X. Deng, Complexity Issues in Bilevel Linear Programming, *Kluwer Acedemic Publishers*, (1998), pp. 149–164.

[6] P. C. Fourie and A. A. Groenwold, The particle swarm optimization algorithm in size and shape optimization., *Structural and Multidisciplinary Optimization*,**23** (2002), 259–267.

[7] J. H. Holland, *Adaptation in natural and artificial systems*, Ann Arbor: Ann Arbor University Press., (1975)

[8] V. Hooke and T.A. Jeeves, Direct search solution of numerical and statistical problems, *Journal of Associated Computation*,**8** (1961), 212–229.

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

[10] W. Z. Lu, H.Y. Fan, A. Y. T. Leung, and J. C. K. Wong, Analysis of pollutant levels in central Hong Kong applying neural network method with particle swarm optimization., *Environmental Monitoring and Assessment*,**79** (2002), 217–230.

[11] W. S. McCulloch and W.A. Pitts, A logical calculus of the ideas immanent in nervous activity, *Bulletin of Mathematics and Biophysics*,**5** (1943), 115–133.

[12] M .S. Osman, W.F., Abd El-Wahad, M. M.K. El Shafei, ,and H.B. Abd El Wahab, A Solution Methodology of Bi-Level Linear Programming Based on Genetic Algorithm, *Journal of Mathematics and Statistics*,**5** (2009), 352–359.

[13] C. O. Ourique, E.C. Biscaia, and J. Carlos Pinto, The use of particle swarm optimization for dynamical analysis in chemical processes., *Computers and Chemical Engineering*,**26** (2002), 1783–1793.
[14] E. I. Papageorgiou, K. E. Parsopoulos, P. P Groumpos and M. N. Vrahatis, , Fuzzy cognitive maps learning through swarm intelligence., *Lecture notes in computer science*, (Vol. 3070, pp. 344–349). Berlin: Springer.
[15] K. E. Parsopoulos and M.N. Vrahatis, On the Computation of all global minimizers through particle swarm optimization., *IEEE Transactions on Evolutionary Computation*,**8** (2004), 211–224.
[16] K. E. Parsopoulos and M. N. Vrahatis, Recent approaches to global optimization problems through particle swarm optimization, *Natural Computing*,**1** (2002c), 235–306.
[17] Y.G Petalas, K.E. Parsopoulos, and M.N. Vrahatis, Memetic particle swarm optimization, *Annals of Operating Research*,**156** (2007), 99–127.
[18] T, Ray and K.M. Liew, A swarm metaphor for multiobjective design optimization., *Engineering Optimization*, **34(2)** (2002), 141–153.
[19] A. Saldam, I. Ahmad and S. Al-Madani, Particle swarm optimization for task assignment problem., *Microprocessors and Microsystems*,**26** (2002), 363–371.
[20] Santo, Isabel A. C. P. Espirito, and Fernandes, Edite M. G. P., Heuristic Pattern Search for Bound Constrained Minimax Problems, *Computational Sciences and Its Applications*, **6784** (2011), 174–184.
[21] H. -P. Schwefel, *Evolution and optimum seeking.* New York:Wiley, (1995)
[22] R. Storn and K. Price, Differential evolutiona simple and efficient heuristic for global optimization over continuous spaces., *Journal of Global Optimization*,**11** (1997), 341-359.
[23] G. Wang, Z. Wan, and X. Wang, Solving Method for a Class of Bilevel Linear Programming based on Genetic Algorithms, *Mathematic Programming Society*,(2003).

Mohamed A. Tawhid got his PhD in Applied Mathematics from the University of Maryland Baltimore County, Maryland, USA. From 2000 to 2002, he was a Postdoctoral Fellow at the Faculty of Management, McGill University, Montreal, Quebec, Canada. Currently, he is a full professor at Thompson Rivers University. His research interests include nonlinear/stochastic/heuristic optimization, operations research, modelling and simulation, data analysis, and wireless sensor network. He has published in journals such as Computational Optimization and Applications, J. Optimization and Engineering, Journal of Optimization Theory and Applications, European Journal of Operational Research, Journal of Industrial and Management Optimization, Journal Applied Mathematics and Computation, etc. Mohamed Tawhid published more than 40 referred papers and edited 5 special issues in J. Optimization and Engineering (Springer), J. Abstract and Applied Analysis, J. Advanced Modeling and Optimization, and International Journal of Distributed Sensor Networks. Also, he has served on editorial board several journals. Also, he has worked on several industrial projects in BC, Canada.

Garret Pauluck Received the B.Sc. degree in computer science and mathematics from Thompson Rivers University. He is currently doing his graduate studies at Department of Mathematics, Simon Fraser University, BC Canada.