Binary Hybrid Particle Swarm Optimization with Wavelet Mutation

Quang Anh Tran[1], Quang Minh Ha[2], Quan Dang Dinh[3], and Frank Jiang[4]

[1] Faculty of Information Technology, Hanoi University, Vietnam
anhtq@hanu.edu.vn
[2] Faculty of Information Technology, Hanoi University, Vietnam
minhhq_fit@hanu.edu.vn
[3] Faculty of Information Technology, Hanoi University, Vietnam
quandd.vnip@gmail.com
[4] School of Engineering and IT, University of New South Wales, Canberra, Australia
F.Jiang@adfa.edu.au

Abstract. Particle swarm optimization (PSO) is an evolutionary algorithm in which individuals, called particles, move around a multidimensional problem space at different directions (trajectories) and speeds (velocities) to find the best solution. A particle movement is based on its previous best result and the previous best result of the entire population. In one of the current PSO variants - HPSOWM [8], a mutation process based on wavelet theory was added to the original PSO to prevent premature conclusion of the best solution. This hybrid PSO has improved solution stability and quality over the original algorithm as well as many other hybrid PSO algorithms. However, it is limited to work on a continuous problem space. In this research, we propose a reworked version of HPSOWM which operates on binary-based problem space, and it is called "Binary Hybrid Particle Swarm Optimization with Wavelet Mutation" (BHPSOWM). In this version, the movement mechanism of particles as well as the mutation process are modified and transformed to work with binary element. We conducted an experiment which compare the performance of the binary version of three algorithms: Genetic Algorithm (GA), Particle Swarm Optimization (BPSO) and our proposed algorithm - BHPSOWM. The result showed that our proposed model deliver better performance in terms of the mean cost value, standard deviation and the convergence rate under the same settings.

Keywords: particle swarm optimization, genetic algorithms, binary

1 Introduction

Among metaheuristic methods, Genetic Algorithm was the first algorithm that utilizes the power of biological mechanisms, and more specifically, the power of natural selection and evolution. It was firstly introduced by Holland in the 1970s [4]. In this algorithm, starting from an initial population of chromosomes (which are bits that compose the program), a proportion of the best of them

are selected, based on a fitness function, to create a new population. Those selected chromosomes are then enter a reproduction (or recombination) process where more than one of them are used to produce a new child solution. The children are then chosen with probability for mutation process, where one or some genes in a chromosome are changed from its initial state to maintain the genetic diversity. Finally we have a new population and the process continues until we reach the optimal solution or time/memory limits.

Genetic Algorithms can provide a good approximation of the solution of all types of problems and it has been successfully applied in many problems including scheduling, optimization of networks, engineering, rule discovery ... [11]

Where GA was influenced by the biological mechanisms, another evolutionary algorithm called Particle Swarm Optimization (PSO) which is introduced by Kennedy and Eberhard in 1995 [6], simulates the the movement of organisms in a bird flock or fish school. It was initially used as an optimization technique for use in real-number problem spaces [7]. Since its creation, there have been many variants which were proposed. They mostly focus on tuning the velocity of the particle, to balance between local search and global search [12][1]. In 1997, Kennedy and Eberhart modified PSO into a binary search space version. They tested with De Jong's five test functions [2] and concludes that binary PSO is capable of solving these problems rapidly.

In this research, we adapt many ideas from binary PSO implementation to propose a binary version of Hybrid PSO with Wavelet Mutation (HPSOWM), which is the work of [8] to solve continuous search space. When HPSOWM was proven to outweigh other approaches (HPSOM [3], HGAPSO [5], HGPSO [10]) in finding optimal solution, convergence rate ..., it hasn't been tested in binary version yet. That is the purpose of this research, to implement binary version of HPSOWM, to evaluate its performance against GA and binary PSO (BSPO) using many different benchmark test functions which are divided into three categories of different characteristics.

This paper is organized as follows: the first section gives the introduction. The second section discusses the backgrounds of this research which are PSO and HPSOWM. Next we discuss about how to transform those two algorithms from continuous to binary version. After that, we describe the experiment with settings, benchmark functions and analyze the results. Finally a conclusion is given with the overall result of this proposed model.

2 Backgrounds

2.1 Particle Swarm Optimization (PSO)

PSO is an algorithm that simulates the sociological behavior of bird flocking. In the search space, we define a swarm X, which has many particles $x^p(t) \in \{x^1(t), x^2(t), \ldots, \gamma\}$ where γ is a number of particles and t is the tth iteration among T total iterations. Each particle x^p consists of many elements, x_i^p where $i = 1, 2, 3, \ldots, \kappa$, with κ is the dimension of a particle.

Firstly, the particles of the swarm are initialized and evaluated by a predefined fitness function. The objective of the PSO is to minimize these fitness function values. In order to move the particle in the search space, a velocity $v_j^p(t)$ is defined. It is the flight speed of the jth element of the pth particle at the tth iteration. Additionally, $x_j^p(t)$ is the current position of that element. The velocity and current position is then calculated by the following formulae:

$$v_j^p(t) = 2 \cdot rand() \cdot (pbest_j^p - x_j^p(t-1)) + 2 \cdot rand() \cdot (gbest_j - x_j^p(t-1)) \quad (1)$$

$$x_j^p(t) = x_j^p(t-1) + v_j^p(t) \quad (2)$$

Another improved version of PSO which is introduced in [**?**], added two factors: the constriction and inertia which change the velocity calculation to:

$$v_j^p(t) = k \cdot \{w \cdot v_j^p(t-1) + \varphi_1 \cdot rand() \cdot (pbest_j^p - x_j^p(t-1)) + \varphi_2 \cdot rand() \cdot (gbest_j - x_j^p(t-1))\} \quad (3)$$

where k is the constriction factor and w is the inertia factor, φ_1 and φ_2 are constants.

Even with the improved version of PSO, when the particle coincides with the global best position, the particle will move away from this point if the two factors above are different from zero. Furthermore, the particles will stop moving when their velocities are close to zero and they catch up with the global best position. This phenomena is called *stagination* [**?**]. To overcome this phenomena, the mutation process of GA has been added to PSO to form up a Hybrid PSO with Mutation (HPSOM). We will discuss this mutation process among with the Wavelet Mutation in the next section.

2.2 Hybrid PSO with Wavelet Mutation

The initial version of PSO with mutation process works as follows: a random particle is selected to be moved to another position in the search space by using mutation under the following operation:

$$mut(x_j) = x_j - \omega, r < 0 \quad (4)$$

$$mut(x_j) = x_j + \omega, r \geq 0 \quad (5)$$

where x_j is a randomly chosen element of the particle above, ω is randomly generated in the range of $[0, 0.1x(para_{max}^j - para_{min}^j)]$, r is a random number between +1 and -1, $para_{max}^j$ and $para_{min}^j$ are the upper and lower bounds of each particle element. Therefore, we can see that ω represents 1/10 of the search space.

This version of PSO can overcome the *stagination* phenomena. However, it can easily be seen that the mutating space is fixed by ω. It is not always the best strategy for using the same size mutation space all the time. Therefore, in [8], the author proposed an improvement method to dynamically adjust the mutating size ω based on the wavelet theory as follows.

Hybrid PSO with Wavelet Mutation Each particle element has a chance to mutate which is controlled by a probability of mutation $p_m \in [0, 1]$. For each element, we generate a random number between 0 and 1 r, if $r \leq p_m$, the mutation will take place on that element. In details, let $x^p(t) = [x_1^p(t), x_2^p(t), \ldots, x_k^p(t)]$ be the current selected particle, with $x_j^p(t)$ is its randomly chosen element, $x_j^p(t) \in [param_{min}^j, param_{min}^j]$. After the mutation process, the resulting particle is $\bar{x}^p(t)$

$$\bar{x}^p(t) = \begin{cases} x_j^p(t) + \sigma \times (para_{max}^j - x_j^p(t)) \text{ if } \sigma > 0 \\ x_j^p(t) + \sigma \times (x_j^p(t) - para_{min}^j) \text{ if } \sigma \leq 0. \end{cases}$$

where $j \in 1, 2, 3, \ldots k$ is the dimension of the particle and:

$$\sigma = \frac{1}{\sqrt{a}} \psi(\frac{\varphi}{a}) \tag{6}$$

where $\psi(x)$ is a wavelet - a continuous time function (see Figure 1) represents a certain seismic signal that satisfies two properties:

$$\int_{-\infty}^{+\infty} \psi(x) dx = 0 \tag{7}$$

and

$$\int_{-\infty}^{+\infty} |\psi(x)|^2 dx < 0 \tag{8}$$

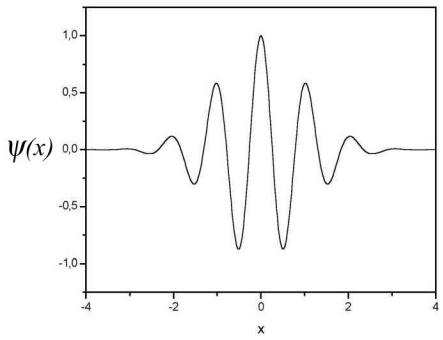


Fig. 1. The original form of the Morlet wavelet

In this paper, we adapt this approach (HPSOWM) and transform the real-valued particles into binary-based ones. This will be discussed in the next section.

3 From PSO, HPSOWM to BPSO, BHPSOWM

A binary-based particle consists of a bit string in which each element can take a value of one or zero. Such particle "moves" by flipping a number of its bits from 0 to 1 and vice versa. A velocity vector composed of bits could be used to modify a binary particle. However, applying formula (**??**) in a binary space by replacing v^p, $pbest^p$ and $gbest$ with bit strings would actually make the particle move randomly around the problem space, not following any direction. The reason for this is that flipping a bit from 0 to 1 means taking a full jump from the lower bound to the upper bound.

What would be the velocity for a binary-based particle? How trajectory should be understood in a binary problem space? These two questions can be answered by defining the values in the velocity vector as "*probabilities that a bit will take value 1 or 0*". Such probability value is used in the following way:

```
if (rand() < probability) then
    x = value1;
else
    x = value2;
```

Note that the $rand()$ function returns a real number x such that $0 \leq x < 1$.

Thus, the velocity vector v^p must be scaled down to real numbers in the range $[0.0, 1.0]$ and x_j^p will take either 0 or 1 as value. Specifically, the velocity value, which is between 0.0 and 1.0, will be compared to a random number in the same range. If it is higher than the random value,x_j^p will take value 1, otherwise x_j^p will take value 0. Note that in the original version of PSO, the magnitudes on the dimensions of velocity vector are limited by their dynamic range, which can be quite large. The rate at which the velocity increases or decreases is also high. To address this problem, a logistic function – the sigmoid function – was used to scale down the velocity values. The function is as follow:

$$S(t) = \frac{1}{(1 + e^{-t})} \tag{9}$$

With the sigmoid function, we can keep the old velocity update formula (**??**). This function makes sure it generates a number between 0.0 and 1.0 from any real number t. This property eliminates the need of a v^{max} value which was used in the original version to limit the particle speed to about one tenth of the search space's length. With the new definition of velocity and the sigmoid function, now particles are updated using the following logic instead of formula (**??**):

$$x_j^p = \begin{cases} 1, & rand() < S(v_j^p) \\ 0, & rand() > S(v_j^p) \end{cases} \tag{10}$$

This transformation makes the particles move more slowly on each dimension towards the two best values. As a result, one bit would change from 0 to 1 before another does, whereas previously, many bits would flip at the same time due to

using a bit string as a velocity vector. This approach allows the particles to scan the problem space much more carefully, reducing the risk of "stepping over" a good solution.

Trajectory in a binary space can now be explained. In a continuous space, the direction of the velocity vector decides the particle's trajectory. As a particle moves closer to the global best position, its coordinates get closer to the coordinates of *gbest*. In a binary space, the "position" of a particle is actually defined by the velocity vector, which contains probabilities. A particle is said to be close to another particle when the probabilities of corresponding bits are close to each other. One interesting thing about this algorithm is that, even when two particles are at the same position they could result in different bit strings thanks to the random value in the particle update process described in (10). Note that with a probability value of 0.9, there is still a 10% chance that a bit would be 0.

4 Experiments & Results

4.1 Benchmark functions

In this paper, we compare the performance of the binary version of GA, PSO and HPSOWM that will be called BGA, BPSO, BHPSOWM respectively, using a suite of benchmark test functions (shown in 4.1). Many different kinds of optimization problems are covered by these benchmark test functions [8]. They are divided into three categories as follows:

- Category I - Unimodal functions: a symmetry model with a single minimum. Function f_1 to f_7 are in this category
- Category II - Multimodal functions with few local minima. Function f_8 to f_{13} belong to this.
- Category III - Multimodal functions with many local minima which includes function f_{14} to f_{18}

Table 1. Benchmark test functions

Test function	Domain range	Optimal point
$f_1(x) = \sum_{i=1}^{30} {x_i}^2$	$-100 \leq x_i \leq 100$	$f_i(0) = 0$
$f_2(x) = \sum_{i=1}^{9} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$	$-2.048 \leq x_i \leq 2.048$	$f_2(1) = 0$
$f_3(x) = \sum_{i=1}^{100} (\lvert x_i + 0.5\rvert)^2$	$-10 \leq x_i \leq 10$	$f_3(0) = 0$
$f_4(x) = \sum_{i=1}^{10} i x_i^4 + random[0, 1)$	$-2.56 \leq x_i \leq 2.56$	$f_4(0) = 0$
$f_5(x) = max\lvert x_i\rvert, 1 \leq i \leq 30$	$-100 \leq x_i \leq 100$	$f_5(0) = 0$
$f_6(x) = \sum_{i=1}^{30} \lvert x_i\rvert + \prod_{i=1}^{30} \lvert x_i\rvert$	$-10 \leq x_i \leq 10$	$f_5(0) = 0$
$f_7(x) = -cos(x_1) \cdot cos(x_2) \cdot exp\Big(-\Big((x_1-\pi)^2+(x_2+\pi)^2\Big)\Big)$	$-300 \leq x_1, x_2 \leq 300$	$f_7(\Big[\pi, \pi\Big]) = -1$
$f_8(x) = \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j+\sum_{i=1}^{2}(x_i - a_{ij})^6}\right]^{-1}$	$-65.536 \leq x_1, x_2 \leq 65.536$	$f_8(\lvert -32, 32\rvert) \approx 1$
$f_9(x) = \sum_{i=1}^{9} \left[a_i - \frac{x_1(b_i^2+b_i x_2)}{b_i^2+b_i x_3+x_4}\right]^2$	$-5 \leq x_i \leq 5$	$f_9(0.1928, 0.1928, 0.1231, 0.1358) \approx 0.0003075$
$f_{10}(x) = -\frac{sin(x_1)sin(x_2)}{x_1 x_2}$	$-10 \leq x_1, x_2 \leq 10$	$\lim_{x \to \lvert 0,0\rvert} f_{10}(x) = -1$
$f_{11}(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$	$-5 \leq x_1, x_2 \leq 5$	$f_{11}(\Big[0.08983, -0.7126\Big]) = f_{11}(\Big[-0.08983, 0.7126\Big]) \approx -1.0316$
$f_{12}(x) = -\sum_{i=1}^{30} c_i exp\left[-\sum_{j=1}^{3} a_{ij}\Big(x_j - p_{ij}\Big)^2\right]$	$0 \leq x_i \leq 1$	$f_{12}(0.114, 0.556, 0.852) \approx -3.8628$
$f_{13}(x) = -\sum_{i=1}^{4} c_i exp\left[-\sum_{j=1}^{6} a_{ij}\Big(x_j - p_{ij}\Big)^2\right]$	$0 \leq x_i \leq 1$	$f_{13}(\Big[0.201, 0.15, 0.477, 0.275, 0.311, 0.62$ -3.32
$f_{14}(x) = 0.1\left\{\begin{array}{l} sin^2(\pi 3x_1) \\ +\sum_{i=1}^{29}(x_i-1)^2 \cdot \Big[1 + sin^2(3\pi x_{i+1})\Big] \\ +(x_3 0 - 1)^2\Big[1 + sin^2(2\pi x_3 0)\Big] \end{array}\right\} + \sum_{i=1}^{30} u(x_i, 5, 100, 4)$	$-50 \leq x_i \leq 50$	$f_{14}(1) = 0$
$f_{15}(x) = \sum_{i=1}^{30}\left[x_i^2 - 10cos(2\pi x_i) + 10\right]$	$-50 \leq x_i \leq 50$	$f_{15}(0) = 0$
$f_{16}(x) = \frac{1}{4000}\sum_{i=1}^{30} x_i^2 - \prod_{i=1}^{30} cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$-600 \leq x_i \leq 600$	$f_{16}(0) = 0$
$f_{17}(x) = -20exp\left(-0.2\sqrt{\frac{1}{30}\sum_{i=1}^{30} x_i^2}\right) - exp\left(\frac{1}{30}\sum_{i=1}^{30} cos2\pi x_i\right) + 20 + e$	$-32 \leq x_i \leq 32$	$f_{17}(0) = 0$
$f_{18}(x) = -\sum_{i=1}^{10}\left(x_i sin\left(\sqrt{\lvert x_i\rvert}\right)\right)$	$-500 \leq x_i \leq 500$	$f_{18}([420.9687, \cdots 420.9687]) = -10 \times 418.9829 = -4189.829$

4.2 Experiment settings

We adapt the settings from the original research of HPSOWM [8] which has some main settings are:

- Swarm size: 50
- Number of runs: 50
- Time limit for each run: 120 seconds
- The probability of crossover for GA: 0.001
- The initial population: we start from a single initial population so we can compare the performance among three algorithms starting from the same point

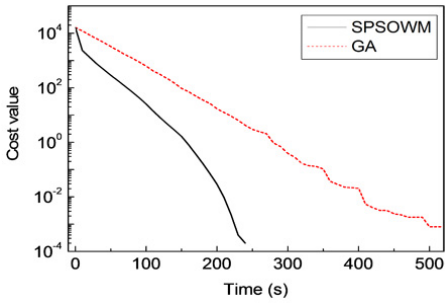
As of the binary version, we also adapt the settings from [7] as:

- V_{max} which is the maximum value for the velocity is set to 6

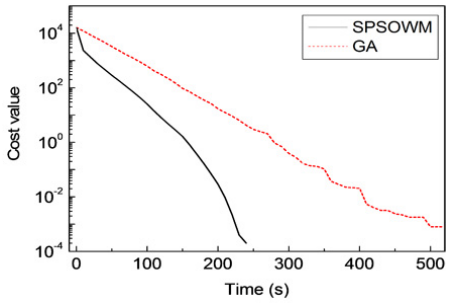
4.3 Results and Discussions

- Compare BHPSOWM with BPSO and GA
- Use benchmark functions in three categories: unimodal functions, multimodal functions with few local minima, multimodal functions with many local minima
- Experiment settings
- Show convergence table
- Show result table with Standard Deviation, ...

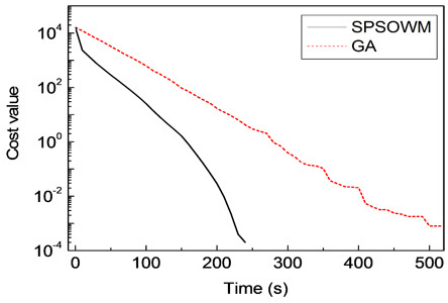
Table 2. Comparison between BHPSOWM, BPSO and GA - Category I: unimodal functions



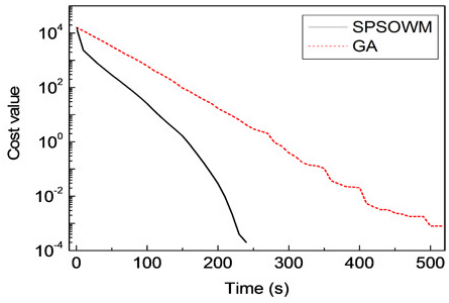
$f_1(x)$



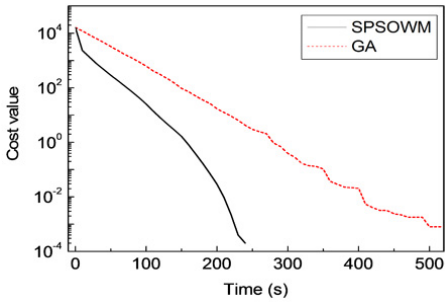
$f_2(x)$



$f_3(x)$

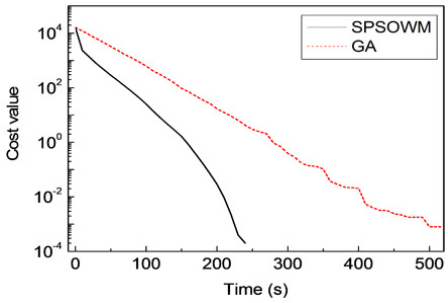


$f_4(x)$

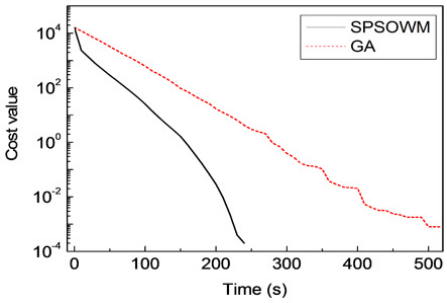


$f_5(x)$

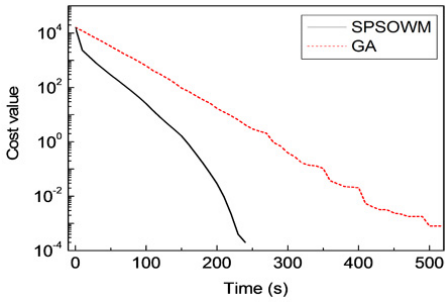
Table 3. Comparison between BHPSOWM, BPSO and GA - Category II: multimodal functions with few local minima



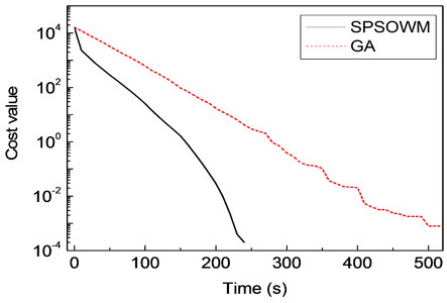
$f_1(x)$



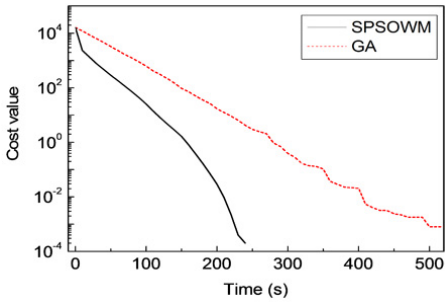
$f_2(x)$



$f_3(x)$

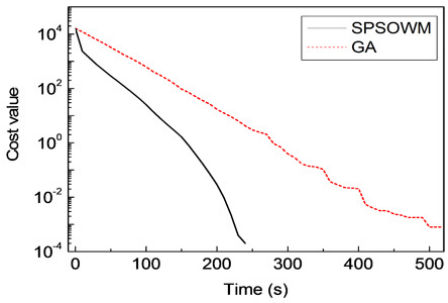


$f_4(x)$

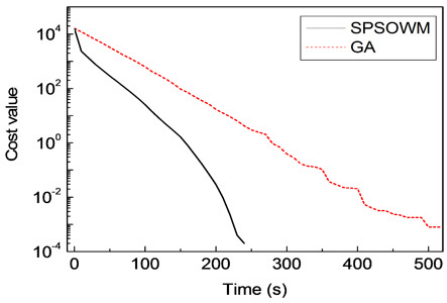


$f_5(x)$

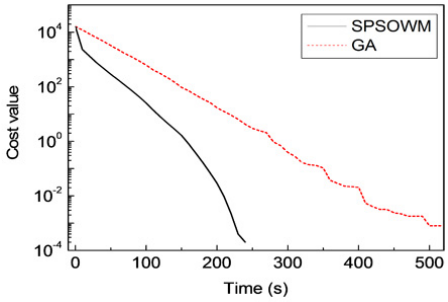
Table 4. Comparison between BHPSOWM, BPSO and GA - Category III: multimodal functions with many local minima



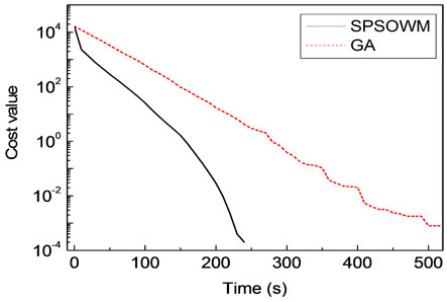
$f_1(x)$



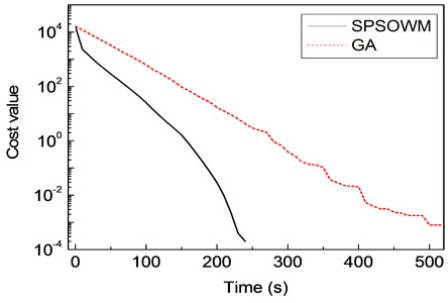
$f_2(x)$



$f_3(x)$



$f_4(x)$



$f_5(x)$

Table 5. Comparison between BHPSOWM, BPSO and GA with benchmark testing function - Category I

		BHPSOWM	BPSO	GA
$f_1(x)$ $T = 120$	Best	0.000	0.000	0.000
	Mean	0.000	0.000	0.000
	Std Dev	0.000	0.000	0.000
$f_2(x)$ $T = 120$	Best	0.000	0.000	0.000
	Mean	0.000	0.000	0.000
	Std Dev	0.000	0.000	0.000
$f_3(x)$ $T = 120$	Best	0.000	0.000	0.000
	Mean	0.000	0.000	0.000
	Std Dev	0.000	0.000	0.000
$f_4(x)$ $T = 120$	Best	0.000	0.000	0.000
	Mean	0.000	0.000	0.000
	Std Dev	0.000	0.000	0.000
$f_5(x)$ $T = 120$	Best	0.000	0.000	0.000
	Mean	0.000	0.000	0.000
	Std Dev	0.000	0.000	0.000
$f_6(x)$ $T = 120$	Best	0.000	0.000	0.000
	Mean	0.000	0.000	0.000
	Std Dev	0.000	0.000	0.000
$f_7(x)$ $T = 120$	Best	0.000	0.000	0.000
	Mean	0.000	0.000	0.000
	Std Dev	0.000	0.000	0.000

Table 6. Comparison between BHPSOWM, BPSO and GA with benchmark testing function - Category II

		BHPSOWM	BPSO	GA
$f_8(x)$ $T = 120$	Best	0.000	0.000	0.000
	Mean	0.000	0.000	0.000
	Std Dev	0.000	0.000	0.000
$f_9(x)$ $T = 120$	Best	0.000	0.000	0.000
	Mean	0.000	0.000	0.000
	Std Dev	0.000	0.000	0.000
$f_10(x)$ $T = 120$	Best	0.000	0.000	0.000
	Mean	0.000	0.000	0.000
	Std Dev	0.000	0.000	0.000
$f_11(x)$ $T = 120$	Best	0.000	0.000	0.000
	Mean	0.000	0.000	0.000
	Std Dev	0.000	0.000	0.000
$f_12(x)$ $T = 120$	Best	0.000	0.000	0.000
	Mean	0.000	0.000	0.000
	Std Dev	0.000	0.000	0.000
$f_13(x)$ $T = 120$	Best	0.000	0.000	0.000
	Mean	0.000	0.000	0.000
	Std Dev	0.000	0.000	0.000

Table 7. Comparison between BHPSOWM, BPSO and GA with benchmark testing function - Category III

		BHPSOWM	BPSO	GA
$f_14(x)$ $T = 120$	Best	0.000	0.000	0.000
	Mean	0.000	0.000	0.000
	Std Dev	0.000	0.000	0.000
$f_15(x)$ $T = 120$	Best	0.000	0.000	0.000
	Mean	0.000	0.000	0.000
	Std Dev	0.000	0.000	0.000
$f_16(x)$ $T = 120$	Best	0.000	0.000	0.000
	Mean	0.000	0.000	0.000
	Std Dev	0.000	0.000	0.000
$f_17(x)$ $T = 120$	Best	0.000	0.000	0.000
	Mean	0.000	0.000	0.000
	Std Dev	0.000	0.000	0.000
$f_18(x)$ $T = 120$	Best	0.000	0.000	0.000
	Mean	0.000	0.000	0.000
	Std Dev	0.000	0.000	0.000

5 Conclusions

6 Acknowledgements

This research was fully supported by the Vietnam National Foundation for Science and Technology Development (NAFOSTED) under project number 102.01-2012.04

Bibliography

[1] Clerc, M., Kennedy, J.: The particle swarm-explosion, stability, and convergence in a multidimensional complex space. Evolutionary Computation, IEEE Transactions on 6(1), 58–73 (2002)

[2] De Jong, K.A.: Analysis of the behavior of a class of genetic adaptive systems (1975)

[3] Esmin, A.A., Lambert-Torres, G., de Souza, A.Z.: A hybrid particle swarm optimization applied to loss power minimization. Power Systems, IEEE Transactions on 20(2), 859–866 (2005)

[4] Holland, J.H.: Genetic algorithms. Scientific american 267(1), 66–72 (1992)

[5] Juang, C.F.: A hybrid of genetic algorithm and particle swarm optimization for recurrent network design. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 34(2), 997–1006 (2004)

[6] Kennedy, J.: Particle swarm optimization. In: Encyclopedia of Machine Learning, pp. 760–766. Springer (2010)

[7] Kennedy, J., Eberhart, R.C.: A discrete binary version of the particle swarm algorithm. In: Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on. vol. 5, pp. 4104–4108. IEEE (1997)

[8] Ling, S.H., Iu, H.H., Chan, K.Y., Lam, H.K., Yeung, B.C., Leung, F.H.: Hybrid particle swarm optimization with wavelet mutation and its industrial applications. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 38(3), 743–763 (2008)

[9] Michalewicz, Z.: Genetic algorithms+ data structures= evolution programs. springer (1996)

[10] Noel, M.M., Jannett, T.C.: Simulation of a new hybrid particle swarm optimization algorithm. In: System Theory, 2004. Proceedings of the Thirty-Sixth Southeastern Symposium on. pp. 150–153. IEEE (2004)

[11] Ross, P., Corne, D.: Applications of genetic algorithms. AISB Quaterly on Evolutionary Computation 89, 23–30 (1994)

[12] Shi, Y., Eberhart, R.: A modified particle swarm optimizer. In: Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on. pp. 69–73. IEEE (1998)

[13] Yao, X., Liu, Y., Lin, G.: Evolutionary programming made faster. Evolutionary Computation, IEEE Transactions on 3(2), 82–102 (1999)