Solving Large Scale Global Optimization Using Improved Particle Swarm Optimizer

Sheng-Ta Hsieh, Tsung-Ying Sun, *Member*, *IEEE*, Chan-Cheng Liu and Shang-Jeng Tsai

***Abstract*—As more and more real-world optimization problems become increasingly complex, algorithms with more capable optimizations are also increasing in demand. For solving large scale global optimization problems, this paper presents a variation on the traditional PSO algorithm, called the Efficient Population Utilization Strategy for Particle Swarm Optimizer (EPUS-PSO). This is achieved by using variable particles in swarms to enhance the searching ability and drive particles more efficiently. Moreover, sharing principals are constructed to stop particles from falling into the local minimum and make the global optimal solution easier found by particles. Experiments were conducted on 7 CEC 2008 test functions to present solution searching ability of the proposed method.**

I. Introduction

PARTICLE swarm optimization (PSO) is a population-based self-adaptive search optimization technique that was proposed by Kennedy and Eberhart [1] in 1995, where the population is referred to as a *swarm*. The PSO is based on simulations of social behaviors such as fish in a school, birds in a flock etc. A swarm in PSO consists of a number of particles. Each particle represents a potential solution of the optimization task. All of the particles iteratively discover a probable solution. Each particle moves to a new position according to the new velocity and the previous positions of the particle. The PSO has ability of fast convergence to local and/or global optimal solutions over a small number of generations.

Similar to other population-based optimization methods such as genetic algorithms (GA), PSO algorithm starts with a random population initialization of particles in the search space [2]. Unlike other evolutionary optimization methods, particles in PSO do not recombine genetic material directly between individuals during the search, but works under social behavior in swarms instead. Therefore, it finds the global best solution by simply adjusting the moving vector of each individual according to personal best and the global best positions of particles in the entire swarm at each time step (generation), in other words, the search process allows particles to stochastically return toward previously successful regions in the search space. The PSO method is becoming more popular due to its simplicity to implement and its ability to quickly converge to a reasonably acceptable solution on a plethora of problems in science and engineering. However, due to the particles' searching behavior, all particles' movement will also refer to global best positions. One the global best positions locate on local minimum. Other particles in the swam may also be trapped into while solving complex multimodal problems [3][4].

For the sake of improving PSO's searching ability and efficiency, this paper presents the efficiency population utilization strategy for particle swarm optimizer (EPUS-PSO). EPUS-PSO is incorporated with the population manager to eliminate redundant particles and hire new ones or maintain particle numbers according to the solution searching status to make the process more efficient. The proposed method also has two built-in sharing strategies based on the sharing principle, which can prevent the solutions from falling into the local minimum and enhance the proposed approach's searching ability.

II. Particle Swarm Optimization (PSO)

In the PSO, each particle moves to a new position according to the new velocity and the previous positions of the particle. This is compared with the best position generated by previous particles in the cost function, and the best one is kept; so each particle accelerates in the direction of not only the local best solution but also the global best position. If a particle discovers a new probable solution, other particles will move closer to it to explore the region more completely in the process [5].

A. Original PSO

Let s denote the swarm size. In general, there are three attributes, current position x_i, current velocity v_i and past best position $Pbest_i$, for particles in the search space to present their features. Each particle in the swarm is iteratively updated according to the aforementioned attributes. Assuming that the function f is the objective function of an optimization problem and the function f need to be minimized, the new velocity of every particle is updated by

$$\begin{aligned} v_{i,j}(g+1) = v_{i,j}(g) + c_1 r_{1_{i,j}}(g)[pbest_{i,j}(g) - x_{i,j}(g)] \\ + c_2 r_{2_{i,j}}(g)[gbest_j(g) - x_{i,j}(g)] \end{aligned} \quad (1)$$

Manuscript received December 12, 2007.

Sheng-Ta Hsieh was with the Department of Electrical Engineering, National Dong Hwa University, Hualien 97401, Taiwan, R.O.C. He is currently with the Department of Communication Engineering, Oriental Institute of Technology, Taipei County, Taiwan, R.O.C. (FO013@mail.oit.edu.tw)

Tsung-Ying Sun, Chan-Cheng Liu and Sheng-Jeng Tsai are with the Department of Electrical Engineering, National Dong Hwa University, Hualien 97401, Taiwan, R.O.C. (E-mail: sunty@mail.ndhu.edu.tw).

978-1-4244-1823-7/08/$25.00©2008 IEEE

For all $j \in 1...N$, $v_{i,j}$ is the velocity of *j*-th dimension of the *i*-th particle, the c_1 and c_2 denote the *acceleration coefficients*, r_1 and r_2 are elements from two uniform random sequences in the range (0, 1), and *g* is the number of generations. The new position of a particle is calculated as follows:

$$x_{i,j}(g+1) = x_{i,j}(g) + v_{i,j}(g+1) \tag{2}$$

The past best position of each particle is updated using

$$pbest_i(g+1) = \begin{cases} pbest_i(g), & \text{if } f(x_i(g+1)) \geq f(pbest_i(g)) \\ x_i(g+1), & \text{if } f(x_i(g+1)) < f(pbest_i(g)) \end{cases} \tag{3}$$

and the global best position *gbest* found from all particles during the previous three steps is defined as:

$$gbest(g+1) = \arg \min_{pbest_i} f(pbest_i(g+1)), \quad 1 \leq i \leq s \tag{4}$$

The value of moving vector v_i can be restricted to the range [-$v_{\max}$, v_{max}] to prevent particles from moving out of the search range

B. Some Variants of PSO

Since Kennedy and Eberhart [1] introduced PSO in 1995, many researchers have worked on improving its performance in various ways. One of the variants introduces a parameter called inertia weight of velocity *w* into the original PSO, introduced by Shi and Eberhart [6]. The new velocity update algorithm is shown as follows:

$$\begin{aligned} v_{i,j}(g+1) = wv_{i,j}(g) + c_1 r_{1_{i,j}}(g)[pbest_{i,j}(g) - x_{i,j}(g)] \\ + c_2 r_{2_{i,j}}(g)[gbest_j(g) - x_{i,j}(g)] \end{aligned} \tag{5}$$

It plays the role of balancing the global search and local search. It can be a positive constant or even a positive linear or nonlinear function of time. This value is typically setup to vary linearly from 1 to 0 during the course of a training run. The inertia weight is similar to the momentum term in a gradient descent neural network training algorithm.

In population-based optimization methods, how the local and global exploration is controlled will influence efficiency on finding the optimal solution directly. PSO has been proven to be effective while applied to optimizing static problems in earlier developments. Even so, most real-world applications are identified as nonlinear dynamic systems. Eberhart and Shi [7] found that the fixed or liner decreased inertia weight for PSO is not very effective for tracking dynamic systems. Considering the dynamic nature of real-world applications, a random inertia weight factor was proposed for tracking dynamic systems instead.

Another interesting approach to improve PSO performance was proposed by Angeline [8] who used a tournament selection process based on the particles' current fitness. This method copies the current positions and velocities of the better half of the population to replace the worse half, without changing the "personal best" values of any of the particles in current step.

The local best version of PSO, *lbest*, reflects the circle neighborhood structure. Particles are influenced by the best position within their neighborhood, as well as their own past experience. While *lbest* is slower in convergence than *gbest*, *lbest* results in much better solutions and searches a larger part of the searching space [9].

More recently, Kennedy investigated other neighborhood topologies, finding that the von Neumann topology resulted in superior performance [10]. In [11], Parsopoulos and Vrahatis proposed a unified particle swarm optimizer (UPSO) which combined both the global version and local version. A cooperative particle swarm optimizer (CPSO) [3] which include two versions: CPSO-S and CPSO-H were also proposed. The CPSO-S model is a direct application of Potter' s CCGA model to the standard PSO, while the CPSO-H model combines the standard PSO with the CPSO-S model, and performs them both in each generation. CPSO split the space (solution vector) into several sub-spaces (smaller vectors) where each sub-space is optimized using a separate swarm, i.e. CPSO use one-dimensional swarms to search each dimension separately, the results are then integrated. Peram *et al.* proposed the fitness-distance-ratio based particle swarm optimization (FDR-PSO) [12], which defines the "neighborhood" of a particle as its *n* closest particles of all particles in the population (measured in Euclidean distance). In [13], the *turn-around factor* was involved in original PSO to solve dynamic problems such as blind source separation. Due to the time-varying optimal solutions in dynamic systems, it may be harder for the particles to catch up to various variations in each time slot, and may produce undesirable results if they move according to previous experiences only. El-Abd and Kamel proposed a hierarchal cooperative particle swarm optimizer [14] which combines two previously proposed models, CONPSO [15] and CPSO-S. The combination is achieved by having two swarms searching for a solution concurrently for solution exchange and performing better convergence. Recently, a comprehensive learning particle swarm optimizer (CLPSO) [4] was proposed. Its learning strategy abandons the global best information, and all other particles' past best information is used to update particles' velocity instead. CLPSO can significantly improve the performance of the original PSO on multimodal problems. There are more and more improved variants of PSO have been proposed, but it is hard to define which variants of PSO is the standard one. Thus, the idea of the standard PSO is to define a real standard at least for one year, validated by some researchers of the field, in particular James Kennedy and Maurice Clerc. This PSO version does not intend to be the best one on the market (in particular there is no adaptation of the swarm size nor of the coefficients) but simply very near of the original version (1995) with just a few improvements based on some recent works [16].

III. EFFICIENCT POPULATION UTILIZATION STRATEGY FOR PARTICLE SWARM OPTIMIZER (EPUS-PSO)

Although PSO algorithms have been applied to a wide range of optimization problems and numerous variants of the PSO exist, solving high complexity problems with efficient searching and rapid convergence are still an active area of research. In order to allow the better particles from exchanging their information, the population manager and solution sharing strategy were introduced to PSO. Also, to improve the particles' searching abilities, an effective searching range sharing strategy was proposed to prevent solutions from falling into the local minimum. The sharing concept, which includes solution sharing and search range sharing, will make particles exchange their experience each other.

A. Population Manager

The first thing to apply the PSO algorithm to solve specific applications is to decide several initial parameters of PSO, which includes a choice of the particle number according to users' experience or complexity of the problem. Having more particles can extend the searching space and increase the probability of finding the global optimal solution, but it will spend more time in each generation. The problem is that, until now, there is no way to know how many particles is suitable for solving the current problem.

Here, a population manager is introduced to EPUS-PSO to enhance its searching ability. The population manager will increase or decrease particle numbers according to the solution searching status, thus the population size in EPUS-PSO is variable. If the particles cannot find a better solution to update the global best solution *gbest*, particles may be trapped into the local minimum during the searching process, or need a competent guide to lead them toward the potential area. Thus, the information (experience) of existing particles may be too little to handle the current solution searching procedures. To keep *gbest* updated and to find better solutions in the current generation, new particles should be added into the swarm and share their up-to-date information to speed up the solution searching progress. In order to avoid unlimited increase in particles, a boundary for population size should be predefined. Therefore, the population manager will adjust population size in following three conditions:

1) If the *gbest* has not been updated in *k* consecutive generations, and if the current population size doesn't exceed the boundary, a new particle will be added into the swarm. The new particle will be created from combining the information of the past best solutions of two randomly selected particles, through a crossover-like information combination strategy to provide some useful information for this particle, meaning, this particle will be placed at a beneficial position to the swarm. The new particle will be involved in the solution searching process in the following generation.

2) If particles can find one or more solutions to update the global best solution *gbest* in *k* consecutive generations, the existing particles may prove to be too many to handle the current solution searching procedures. For saving some time on finding the global optimal solution, and keep *gbest* updated, the redundant particles should be expelled from the swarm PSO to conserve their evolution time for speeding up the solution searching progress. If *gbest* has been updated in two consecutive generations, and the current population size is more than one, a particle with poor performance in the current generation will be removed from the swarm. In the following generation, the particle number will be one less the original.

3) If the particles cannot find any better solutions to update global best solution, but the population size is equal to the boundary, a particle with poor performance in the current generation will be removed to reserve a space to accommodate a new potential particle.

where *k* is the population manager activating threshold which is a positive integer number. While *k* is set as 1, the population size will be adjusted in each generation. Finally, if the *gbest* has been updated intermittently, population size will remain the same. Setting a higher *k* value for the population manager will give the population adjustment lower sensitivity; while a lower *k* value for the population manager will increase its sensitivity so it could better process solution searching states (adjust population on time).

The possibility that the new particle may be eliminated instantly because it was born with a poor solution counteracts the purpose of the population manager. To prevent this situation, the value *k* should set more than 1. This will ensure the new born particle will live for more than one generation. In this paper, the *k* value is set as 2 in order to prevent the instant elimination of a newborn particle, and to keep population manager high sensitivity.

An example of the change in population size is showed in Fig. 1. It's for solving the Quadric function [17] with ten dimensions. The initial population size is set as 1 and the maximum population size is set as 30. Population manager will dynamically adjust the particle numbers according to the current solution searching state.

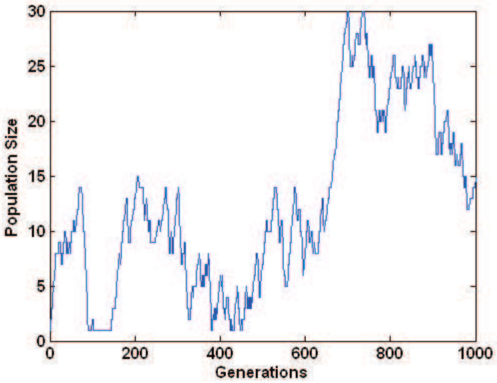


Fig. 1 An example of the change in population size while solving the Quadric function with 30 dimensions.

B. Solution Sharing

In the proposed solution sharing strategy, the following equation was used to update the velocity:

$$v_{i,j}(g+1)=\begin{cases} wv_{i,j}(g)+c_1r_{1_i}(g)[pbest_{i,j}(g)-x_{i,j}(g)] \\ \qquad +c_2r_{2_i}(g)[gbest_j(g)-x_{i,j}(g)] \quad \text{if } rand \ge Ps_i \\ wv_{i,j}(g)+c_1r_{1_i}(g)[pbest_{i,j}(g)-x_{i,j}(g)] \\ \qquad +c_2r_{2_i}(g)[pbest_{r,j}(g)-x_{i,j}(g)] \quad \text{if } rand < Ps_i \end{cases} \tag{6}$$

where Ps_i denotes the solution sharing probability, whose definition will have a detail description later, of the *i*th particle. It will decide if the moving vector of the third item in the velocity update equation is referring to the *gbest* or another particle's *pbest* ($pbest_r$). The $pbest_r$ can be any other particle's *pbest*, not including its own.

In [4], Liang *et al*. found that different Learning Probability values will affect the results for the same problem if the same value of learning probability was used for all the particles in the swarm. Thus, each particle will be given a unique sharing probability. Before the particle's new velocity is calculated, a random number will be generated. If this number is larger than or equal to Ps_i, the guide of the third item of the velocity update equation will be the *gbest*. On the other hand, if this number is smaller than Ps_i, the particle will learn from another particle's *pbest*. In other words, the guide of the third item (*gbest*) of the velocity update equation will be replaced by $pbest_r$. The selection of $pbest_r$ is stated as follows.

If the random number is smaller than Ps_i

1) Randomly choose two particles' *pbest* from the population (Except the $pbest_i$).
2) Compare the fitness values of both *pbest* and select the better one as $pbest_r$.
3) $pbest_r$ will share its own information of all dimensions.

Thus, the solution searching of all the particles refer to not only their own *pbest* but also have the chance for learning from other particles' *pbest*.

Since the dimensional information is incorporated in calculating the particles' solution sharing probability. The definition of learning probability in [4] was referred to define the solution sharing probability for each particle:

$$Ps_i = 0.5\times\frac{(N-1)\times\exp\left(\dfrac{i-1}{s-1}\right)-1}{N} \tag{7}$$

where N denotes the dimension of problems and s is the population size. The solution sharing strategy will be not necessary while solving one-dimensional problems. The solution sharing probability will be set as zero due to $N-1=0$.

There are two main differences between the solution sharing strategy and the comprehensive learning strategy [4].

1) The solution sharing strategy not only learns from other particles' experiences but also refers to *gbest*'s information.
2) For each particle, the solution sharing strategy picks another particle's *pbest* as one of the guides for current movement instead of fine tuning dimensions one by one.

C. Searching Range Sharing (SRS)

According to the searching behavior of PSO, the *gbest* will be an important clue in leading particles to the global optimal solution. But it is unavoidable for the solution to fall into the local minimum while particles try to find better solutions. In fact, after several generations, particles will gather in several clusters, or even just one cluster, which is the local minimum. Each particle in the cluster may perform a local search to follow evolution, but not be able to explore other better solutions.

In order to allow the solution exploration in the area to produce more potential solutions, and find unsearched solution space. The searching range sharing (SRS) strategy, a mutation-like evolutionary strategy, was introduced to the EPUS-PSO algorithm. The SRS strategy can be classified into two versions: local and global. The two versions of SRS are classified according to a restricted boundary. Similar to mutation operation, the SRS activates under a predefined SRS rate which can be setup to vary linearly from 0 to 0.2 during the course of a training run. Note that this is similar to the temperature adjustment schedule found in Simulated Annealing (SA) algorithm.

In the local version, the particle's new position will be restricted in the boundary of all past best solution of all particles ($[pbest_{\min}, pbest_{\max}]$). Through local SRS, the perturbed particles will start solution searching in the reduced range. This will increase efficiency of solution searching for particles. Similarly, in the global version, the particles' new positions will be restricted in the search boundary as an initial state ($[X_{\min}, X_{\max}]$). Through global SRS, the perturbed particles are randomly distributed in the initial search range. It will increase the probability of finding potential solutions in un-searched areas. The local version can share particles' searching ranges to make solution searching for particles more efficient. The global version can prevent solution from being trapped in the local optimum.

TABLE I
SEARCH RANGE SHARING STRATEGY

	Local version	Global version
For Particles	To perturb selected particle and place them in reduced range $[pbest_{\min}, pbest_{\max}]$	To perturb particle and place them in initial range $[X_{\min}, X_{\max}]$
For Dimensions	To perturb selected dimension ($d1$) of current particle to another selected dimension ($d2$)'s searching range $[pbest_{d2_{\min}}, pbest_{d2_{\max}}]$	To Perturb selected dimension ($d1$) of current particle to another selected dimension ($d2$)'s searching range $[X_{\min}, X_{\max}]$

The SRS not only can share searching range between particles but also share searching range between dimensions. For particles, the particles' current positions (solution) for all dimensions will be perturbed. This particle will be placed at a new position in the restricted boundaries while the SRS is activated. For dimensions, a dimension will be picked up randomly; all particles' corresponding dimensions will be perturbed and restricted as another dimension's past best solution for all particles. For example, for a dimension's local SRS, a randomly selected dimension ($d1$) will incorporate another randomly selected dimension ($d2$)'s searching range, the corresponding $d1$ in the particles i will be perturbed in the range between $[pbest_{d2_{\min}}, pbest_{d2_{\max}}]$, where $pbest_{d2_{\min}}$ and $pbest_{d2_{\max}}$ are the minimal and maximal past best solution respectively of corresponding $d2$ of all particles. The dimension's SRS will ignore other dimensions' but reconfigure solutions of dimensions one by one in the particles. The SRS strategy can be summarized in Table I.

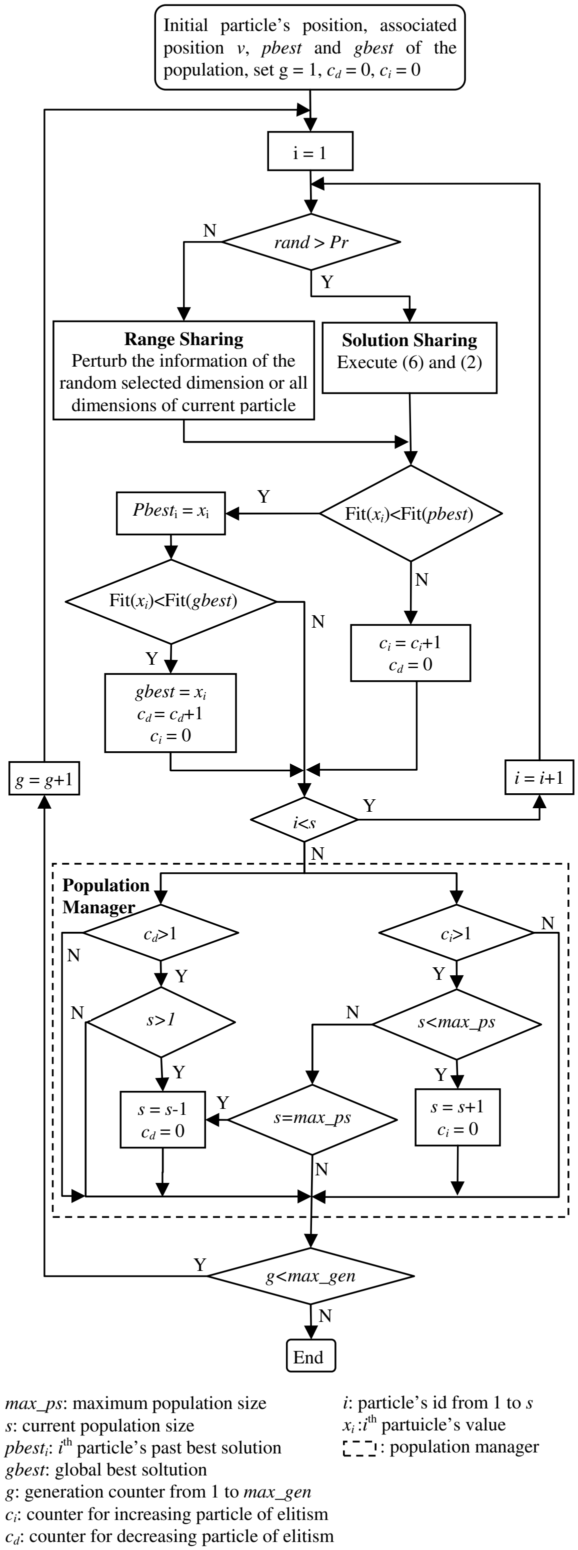
Similar to the mutation rate for of GAs, different SRS rate will affect EPUS-PSO's performance directly. After numerous generations, particles are gathering in several clusters, and therefore the demands of SRS will also be increasing. Due to a lower SRS activating rate may not be able to rescue particles that were trapped into local minimal, and a higher one would interference particles' convergence deeply. To keep better solutions will be found efficiently and also to avoid particles perform local search. The concept of linearly variation inertia weight for PSO is adopted for the SRS rate. The activating probability of SRS for each generation is defined as following:

$$Pr(g) = 0.03 + \frac{0.07 * g}{max\_gen} \tag{8}$$

where g denotes the generation number and max_gen is maximum generations. The probability of SRS will keep increased linearly, from 0.03 to 0.1, while solution searching is in progress. Note that if each dimension has different bounds by strict constraint, the SRS can just share searching range between particles but dimensions.

D. EPUS-PSO Algorithm

The complete flowchart of the EPUS-PSO is given in Fig. 2. After initializing all the parameters of EPUS-PSO, each particle will be conducted by either solution sharing or range sharing strategy. Which sharing strategy will be chosen is dependent on Pr. For each particle i, a random number will be generated for it. If this number is larger than Pr, the sharing strategy will be solution sharing; otherwise it will be range sharing. The fitness evaluation will then update the particle's *pbest* and *gbest*. After all particles in the population have moved, the population manager will determine the elimination of a redundant particle with poor performance, the hiring a new one, or the maintenance of the current population size according to the solution searching status. All particles in the population will perform the next iteration after the population manager. The clamping for restrict particles to stay in the searching space is not necessary duo to the poor particles will be eliminated by the population manager. Note that each update step (particle's movement) in the EPUS-PSO is performed on a full N-dimensional vector. This will save a considerable amount of computational consumption when performing each particle's movement.



Initial particle's position, associated position v, pbest and gbest of the population, set g = 1, c_d = 0, c_i = 0
i = 1
rand > Pr
N
Y
Range Sharing
Perturb the information of the random selected dimension or all dimensions of current particle
Solution Sharing
Execute (6) and (2)
Fit(x_i)<Fit(pbest)
Y
Pbest_i = x_i
N
c_i = c_i+1
c_d = 0
Fit(x_i)<Fit(gbest)
Y
N
gbest = x_i
c_d = c_d+1
c_i = 0
g = g+1
i = i+1
i<s
Y
N
Population Manager
c_d>1
c_i>1
N
Y
s>1
s<max_ps
s=max_ps
s = s-1
c_d = 0
s = s+1
c_i = 0
g<max_gen
Y
N
End
max_ps: maximum population size
s: current population size
pbest_i: i^th particle's past best solution
gbest: global best soltution
g: generation counter from 1 to max_gen
c_i: counter for increasing particle of elitism
c_d: counter for decreasing particle of elitism
i: particle's id from 1 to s
x_i : i^th partuicle's value
: population manager

Fig. 2 Flowchart of the EPUS-PSO.

IV. Experiments

In the experiments, seven CEC 2008 test functions including two unimodal and five multimodal functions was chosen for testing the proposed method. The system environments are listed as follows:

TABLE II
ESTIMATED RUNTIME FOR THE TEST SUITE

System	Windows XP (SP2)
CPU	Intel Core 2 Duo E4400 (2.0 GHz x 2). Only one of the processors was used.
RAM	1 G
Language	Matlab 7.4
Algorithm	EPUS-PSO

Although the system is with a Core 2 Duo processor, for fair comparison, one of the two processors was used, i.e. only 50% computation resource of this CPU was used.

In the experiments, the parameters of the proposed method are listed as follows:

- Inertia weight (w): $w = \frac{1}{2*\ln(2)}$
- Acceleration coefficients (c_1 and c_2): $c_1 = c_2 = 0.5 + \ln(2)$

The experiments of the proposed approaches on the seven test functions with 100, 500 and 1000 dimensions are executed for 500000, 2500000 and 5000000 FES respectively. Table IV, V and VI present the 1st (best), 7th, 13th (median), 19th, 25th (worse), mean, and standard deviation of 25 runs of the proposed method on the 7 test functions with 100, 500 and 1000 dimensions respectively. The median convergence graphs (7th) of the 7 test functions with 1000 dimensions are presented in Fig. 3 and 4.

The estimated runtime for the test suite are listed as follows:

TABLE III
ESTIMATED RUNTIME FOR THE TEST SUITE

Dimensions	1000-D
Problems	Function 1-7
Algorithm	EPUS-PSO
Runs	Only one time
Max_FEs	5,000,000
PC	CPU: Intel Core 2 Duo E4400 (2.0 GHz x 2). Only one of the processors was used. RAM: 1 G
Runtime	6 h 10m 53s

The proposed method spends about 6 hours to execute all the 7 functions with 1000 dimensions for single run.

TABLE IV
ERROR VALUES ACHIEVED FOR PROBLEMS 1-7, WITH D=100

FES	Problems	1	2	3	4	5	6	7
5.00e+3	1st (Best)	1.59e+05	6.30e+01	2.15e+10	1.51e+03	1.35e+03	2.00e+01	-8.92e+02
	7th	1.87e+05	7.53e+01	3.53e+10	1.55e+03	1.61e+03	2.04e+01	-8.19e+02
	13th (Median)	2.15e+05	8.03e+01	4.51e+10	1.60e+03	1.68e+03	2.05e+01	-8.12e+02
	19th	2.19e+05	8.44e+01	5.11e+10	1.64e+03	1.73e+03	2.06e+01	-7.98e+02
	25th (Worst)	2.42e+05	8.82e+01	5.46e+10	1.66e+03	1.94e+03	2.07e+01	-7.84e+02
	Mean	2.07e+05	7.94e+01	4.28e+10	1.60e+03	1.68e+03	2.05e+01	-8.12e+02
	Std	2.14e+04	6.81e+00	9.41e+09	4.85e+01	1.42e+02	1.65e-01	2.25e+01
5.00e+4	1st (Best)	2.81e+03	3.47e+01	3.33e+07	8.14e+02	2.47e+01	1.13e+01	-8.92e+02
	7th	3.76e+03	3.71e+01	5.87e+07	9.48e+02	3.19e+01	1.38e+01	-8.40e+02
	13th (Median)	4.17e+03	3.86e+01	8.24e+07	9.90e+02	3.79e+01	1.52e+01	-8.32e+02
	19th	4.94e+03	4.08e+01	1.39e+08	1.14e+03	4.46e+01	1.72e+01	-8.22e+02
	25th (Worst)	6.81e+03	4.75e+01	2.44e+08	1.47e+03	5.31e+01	2.02e+01	-8.13e+02
	Mean	4.43e+03	3.93e+01	9.87e+07	1.06e+03	3.87e+01	1.55e+01	-8.34e+02
	Std	9.46e+02	3.19e+00	4.84e+07	1.91e+02	8.60e+00	2.43e+00	1.75e+01
5.00e+5	1st (Best)	4.43e-01	1.46e+01	6.34e+02	3.67e+02	2.52e-01	1.04e+00	-8.92e+02
	7th	6.14e-01	1.72e+01	8.63e+02	4.26e+02	3.40e-01	1.78e+00	-8.64e+02
	13th (Median)	7.49e-01	1.85e+01	1.59e+03	4.71e+02	3.73e-01	2.00e+00	-8.53e+02
	19th	8.82e-01	2.01e+01	8.27e+03	5.18e+02	4.04e-01	2.35e+00	-8.43e+02
	25th (Worst)	1.04e+00	2.32e+01	1.68e+04	5.60e+02	5.25e-01	2.90e+00	-8.37e+02
	Mean	7.47e-01	1.86e+01	4.99e+03	4.71e+02	3.72e-01	2.06e+00	-8.55e+02
	Std	1.70e-01	2.26e+00	5.35e+03	5.94e+01	5.60e-02	4.40e-01	1.35e+01

TABLE V
ERROR VALUES ACHIEVED FOR PROBLEMS 1-7, WITH D=500

Problems FES		1	2	3	4	5	6	7
2.50e+4	1st (Best)	1.20e+06	5.42e+01	3.72e+11	7.68e+03	9.76e+03	2.08e+01	-3.48e+03
	7th	1.23e+06	7.08e+01	3.88e+11	8.02e+03	1.01e+04	2.08e+01	-3.42e+03
	13th (Median)	1.28e+06	8.72e+01	4.03e+11	8.32e+03	1.03e+04	2.09e+01	-3.41e+03
	19th	1.30e+06	9.14e+01	4.28e+11	8.49e+03	1.04e+04	2.09e+01	-3.39e+03
	25th (Worst)	1.33e+06	9.48e+01	4.44e+11	8.67e+03	1.07e+04	2.09e+01	-3.36e+03
	Mean	1.27e+06	8.04e+01	4.05e+11	8.26e+03	1.03e+04	2.09e+01	-3.41e+03
	Std	3.78e+04	1.33e+01	2.07e+10	2.85e+02	2.12e+02	3.36e-02	2.97e+01
2.50e+5	1st (Best)	9.06e+04	4.61e+01	6.86e+09	4.71e+03	6.78e+02	2.03e+01	-3.56e+03
	7th	1.03e+05	4.66e+01	7.23e+09	4.98e+03	8.03e+02	2.06e+01	-3.48e+03
	13th (Median)	1.06e+05	4.71e+01	7.98e+09	5.12e+03	8.42e+02	2.07e+01	-3.45e+03
	19th	1.12e+05	4.75e+01	8.96e+09	5.19e+03	8.87e+02	2.08e+01	-3.45e+03
	25th (Worst)	1.21e+05	4.95e+01	1.04e+10	5.40e+03	9.81e+02	2.08e+01	-3.42e+03
	Mean	1.07e+05	4.72e+01	8.10e+09	5.08e+03	8.44e+02	2.07e+01	-3.47e+03
	Std	8.20e+03	8.15e-01	1.00e+09	1.72e+02	6.88e+01	1.42e-01	3.99e+01
2.50e+6	1st (Best)	6.91e+01	4.27e+01	3.81e+04	3.27e+03	1.58e+00	5.66e+00	-3.56e+03
	7th	7.99e+01	4.31e+01	5.23e+04	3.40e+03	1.60e+00	6.39e+00	-3.52e+03
	13th (Median)	8.33e+01	4.35e+01	5.72e+04	3.51e+03	1.63e+00	6.61e+00	-3.50e+03
	19th	8.89e+01	4.40e+01	6.15e+04	3.57e+03	1.66e+00	6.86e+00	-3.50e+03
	25th (Worst)	9.82e+01	4.46e+01	7.93e+04	3.72e+03	1.75e+00	7.81e+00	-3.48e+03
	Mean	8.45e+01	4.35e+01	5.77e+04	3.49e+03	1.64e+00	6.64e+00	-3.51e+03
	Std	6.40e+00	5.51e-01	8.04e+03	1.12e+02	4.69e-02	4.49e-01	2.10e+01

TABLE VI
ERROR VALUES ACHIEVED FOR PROBLEMS 1-7, WITH D=1000

Problems FES		1	2	3	4	5	6	7
5.00e+4	1st (Best)	2.56e+06	5.01e+01	8.37e+11	1.52e+04	2.19e+04	2.10e+01	-6.61e+03
	7th	2.61e+06	5.40e+01	8.72e+11	1.56e+04	2.31e+04	2.10e+01	-6.50e+03
	13th (Median)	2.62e+06	6.10e+01	8.83e+11	1.62e+04	2.34e+04	2.10e+01	-6.48e+03
	19th	2.66e+06	8.34e+01	9.05e+11	1.65e+04	2.37e+04	2.10e+01	-6.47e+03
	25th (Worst)	2.74e+06	9.66e+01	9.26e+11	1.72e+04	2.43e+04	2.10e+01	-6.44e+03
	Mean	2.63e+06	6.79e+01	8.88e+11	1.61e+04	2.34e+04	2.10e+01	-6.49e+03
	Std	4.35e+04	1.65e+01	2.30e+10	6.11e+02	5.01e+02	1.33e-02	3.42e+01
5.00e+5	1st (Best)	3.21e+05	4.75e+01	3.13e+10	1.03e+04	2.82e+03	2.09e+01	-6.62e+03
	7th	3.35e+05	4.79e+01	3.54e+10	1.05e+04	3.00e+03	2.09e+01	-6.57e+03
	13th (Median)	3.44e+05	4.81e+01	3.73e+10	1.06e+04	3.10e+03	2.10e+01	-6.55e+03
	19th	3.57e+05	4.84e+01	3.89e+10	1.07e+04	3.16e+03	2.10e+01	-6.54e+03
	25th (Worst)	3.74e+05	5.77e+01	4.63e+10	1.08e+04	3.32e+03	2.10e+01	-6.51e+03
	Mean	3.46e+05	4.85e+01	3.77e+10	1.06e+04	3.09e+03	2.10e+01	-6.55e+03
	Std	1.48e+04	1.90e+00	3.30e+09	1.42e+02	1.30e+02	1.62e-02	2.59e+01
5.00e+6	1st (Best)	5.07e+02	4.59e+01	5.35e+05	7.27e+03	5.15e+00	1.33e+01	-6.72e+03
	7th	5.26e+02	4.64e+01	7.56e+05	7.48e+03	5.61e+00	1.65e+01	-6.63e+03
	13th (Median)	5.54e+02	4.66e+01	8.22e+05	7.60e+03	5.93e+00	2.03e+01	-6.62e+03
	19th	5.75e+02	4.69e+01	9.39e+05	7.65e+03	6.08e+00	2.08e+01	-6.59e+03
	25th (Worst)	6.14e+02	4.73e+01	1.30e+06	8.01e+03	6.81e+00	2.09e+01	-6.58e+03
	Mean	5.53e+02	4.66e+01	8.37e+05	7.58e+03	5.89e+00	1.89e+01	-6.62e+03
	Std	2.86e+01	4.00e-01	1.52e+05	1.51e+02	3.91e-01	2.49e+00	3.18e+01

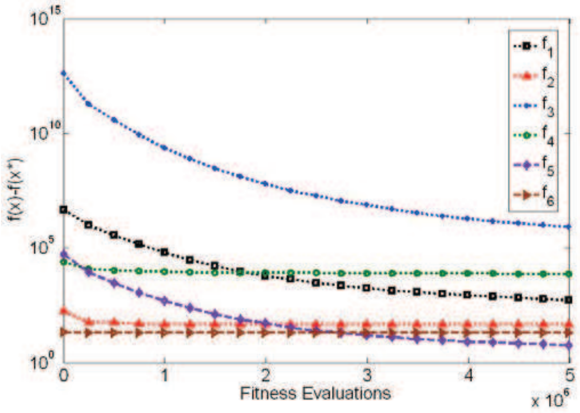


Fig. 3 Convergence Graph for Functions 1-6.

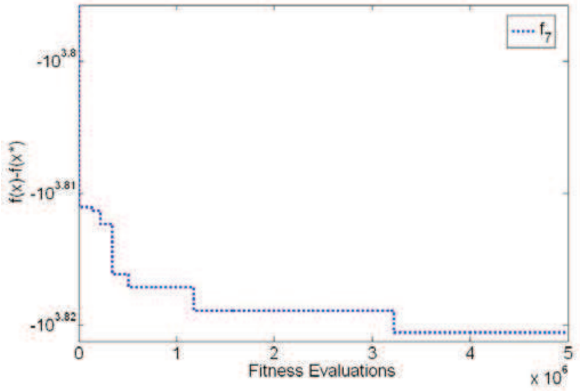


Fig. 4 Convergence Graph for Functions 7.

V. Conclusions

This paper presents an efficiency population utilization strategy for particle swarm optimizer (EPUS-PSO) for solving large scale global optimization which with both unimodal and multimodal problems. The proposed population manager and sharing principle can significantly improve particles' searching abilities, to more easily find the global optimal solution. It also makes PSO more robust, prevents particles from falling into the local minimum, and drives particles more efficiently.

Seven test functions were adopted for testing through a reasonable average and the results are very reliable. Although the EPUS-PSO contains three main parts which differentiates from the original PSO, each of them introduce a simple concept of strategy for particles' movement and adjustment of swarm size. Thus, the EPUS-PSO is simple and easy to implement for solving optimization.

References

[1] R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proc*. 6th Int. Symp. Micro Machine and Human Science, Nagoya, Japan, pp. 39-43, 1995.

[2] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[3] F. van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, pp. 225-239, Jun. 2004.

[4] J. J. Liang, A. K. Qin, P. N. Suganthan and S. Baskar, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions," IEEE Transactions on Evolutionary Computation, vol. 10, Jun. pp. 281-296, 2006.

[5] V. G. Gudise and G. K. Venayagamoorthy, "Comparison of Particle Swarm Optimization and Backpropagation as Training Algorithms for Neural Networks." IEEE Swarm Intelligence Symposium, pp. 110-117, Apr. 2003.

[6] Y. Shi and R. Eberhart, "A modified particle swarm optimizer", in *Proc*. of IEEE World Congress on Computational Intelligence, pp. 69-73, May 1998.

[7] R. C. Eberhart and Y. Shi, "Tracking and optimizing dynamic systems with particle swarms," in *Proc.* IEEE world congress on Evolutionary Computation 2001 (CEC 2001), pp. 94–97, May 2001.

[8] P. Angeline, "Using selection to improve particle swarm optimization," in *Proc.* International joint conference on Neural Networks (IJCNN'99), pp. 84–89, Jul. 1999.

[9] A. P. Engelbrecht, *Computational intelligence an introduction*, Wiley, pp.185-195, 2002.

[10] J. Kennedy and R. Mendes, "Population structure and particle swarm performance," in *Proc.* IEEE world congress on Evolutionary Computation 2001 (CEC 2002), pp. 1671–1676, May 2002.

[11] K. E. Parsopoulos and M. N. Vrahatis, "UPSO—A unified particle swarm optimization scheme," in *Lecture Series on Computational Sciences*, pp. 868–873, 2004.

[12] T. Peram, K. Veeramachaneni, and C. K. Mohan, "Fitness-distance-ratio based particle swarm optimization," in *Proc*. Swarm Intelligence Symposium, pp. 174-181, 2003.

[13] C. L. Lin, S. T. Hsieh, T. Y. Sun and C. C. Liu, "PSO-based learning rate adjustment for blind source separation," in *Proc*. of International Symposium on Intelligent Signal Processing and Communications Systems (ISPACS), pp. 181-184, Dec. 2005.

[14] M. El-Abd and M. S. Kamel, "A Hierarchal Cooperative Particle Swarm Optimizer," in *Proc*. Swarm Intelligence Symposium, pp. 43-47, 2006.

[15] S. Baskar and P. N. Suganthan, "A novel concurrent particle swarm optimization," in Proc. IEEE Congress on Evolutionary Computation, vol. 1, 2004, pp. 792–796.

[16] http://particleswarm.info/