A Novel Particle Swarm Optimization Algorithm

Shahriar Asta[1] and A. Şima Uyar[1],

[1] Computer & Informatics Faculty
Istanbul Technical University, Istanbul, Turkey

{asta, etaner}@itu.edu.tr

Abstract. In this study a novel memory based particle swarm optimization algorithm is presented. This algorithm utilizes external memory. A set of globally found best and worst positions, along with their parameters are stored in two separate external memories. At each iteration, a coefficient, based on the distance of the current particle to the closest best and closest worst particles is calculated. When updating the velocity component, this coefficient is added to the current velocity of the particle with a certain probability. Also randomized upper and lower bound values have been defined for the inertia component. The algorithm is tested on benchmark functions and it is shown empirically that it converges faster to the optima. It also outperforms the PSO and a recent improved PSO, as well as maintaining a superior precision in comparison. Convergence speed is particularly important since the method will be used in a realistic robot motion simulator environment in which the simulation time is long enough to make convergence speed a primary concern.

Keywords: Particle Swarm Optimization, External Memory.

1 Introduction

Particle Swarm Optimization (PSO) is a nature inspired meta-heuristic method. This method was first introduced by Kennedy and Eberhart in 1995 [1]. It is inspired by the swarm behavior of birds flocking, and utilizes this behavior to guide the particles to search for globally optimal solutions.

Basically, in PSO, a population of particles is spread randomly throughout the search space. The particles are assumed to be flying in the search space. The velocity and position of each particle is updated iteratively based on personal and social experiences. Each particle possesses a local memory in which the best so far achieved experience is stored. Also a global memory keeps the best solution found so far. The sizes of both memories are restricted to one. The local memory represents the personal experience of the particle and the global memory represents the social experience of the swarm. The balance between the effect of the personal and social experiences are maintained using randomized correction coefficients. The philosophy behind the velocity update procedure is to reduce the distance between the particle

and the best personal and social known locations. PSO is very easy to implement and there have been many successful implementations in several real world applications.

PSO is a population based heuristic approach. It can get stuck in local optima when dealing with complex multimodal functions. This is why accelerating the convergence speed as well as avoiding the local optima are two primary goals in PSO research. Multiple methods and approaches have been suggested to improve the performance of the original PSO in terms of these goals.

In [2], these efforts have been divided into four categories. The first category includes the parameter selection methods. Introducing inertia and constriction factors into the basic velocity expression or developing strategies for time independent variation of algorithm parameters are among the many methodologies in this category. The method presented in this paper fits best within this category. Other categories given in [2] are: Applications of PSO to different problem areas (second category); Generation of different algorithm strategies and analysis of convergence (third category); Hybridization (fourth category). Although, the novel approach presented in this paper focuses on parameter selection, it also tries to generate a different strategy and attempts to reduce the iteration count. This is why, it can also be included in the third category.

The proposed algorithm utilizes external memory. A set of globally found best and worst positions, along with their parameters are stored in two separate external memories. At each iteration, a coefficient, based on the distance of the current particle to the closest best and closest worst particles is calculated. When updating the velocity component, this coefficient is added to the current velocity of the particle with a certain probability. Also randomized upper and lower bound values have been defined for the inertia component. The algorithm is tested on benchmark functions and it is shown empirically that it converges faster to the optima. It also outperforms the PSO and a recent improved PSO, as well as maintaining a superior precision in comparison. Convergence speed is particularly important since the method will be used in a realistic robot motion simulator environment in which the simulation time is long enough to make convergence speed a primary concern.

This paper is organized as follows. First a brief review of PSO is given (section 2). Then some recent versions of PSO utilizing additional memory are discussed (section 3). The proposed approach is introduced in the subsequent section (section 4). The comparative experimental results are shown in section 5 and finally the conclusion and future works are presented in the last section (section 6).

2 Particle Swarm Optimization

In the basic PSO, each particle is considered as a potential solution to the numerical optimization problem in a D dimensional space. Every particle has a position in this search space and a velocity assigned to it. The position of the particle is represented by $X_i = x_{i1}, x_{i2}, \dots, x_{iD}$. The velocity of a particle is given as $V_i = v_{i1}, v_{i2}, \dots, v_{iD}$.. Also, each particle has a local memory (*pBest*) which keeps the best position that is experienced by the particle so far. A globally shared memory (*gBest*) keeps the best

global position found so far. This information contributes to the flying velocity of each particle, using the following equation:

$$v_i = v_i + \varphi_1 \times rand \times (pBest_i - x_i) + \varphi_2 \times rand \times (gBest - x_i) \quad (1)$$

$$x_i = x_i + v_i \quad (2)$$

where, $\varphi_1$and φ_2 are constants determining the relative influences of the personal and social experiences. Defining an upper bound for the velocity component increases the performance of the approach. Eq.2 gives the particle position update.

In [3], it has been shown that the introduction of an inertia factor to Eq.1 improves performance, since it adjusts the velocity over time and improves the search precision of the particles. Eq.1 can be rewritten as:

$$v_i = \theta \times v_i + \varphi_1 \times rand \times (pBest_i - x_i) + \varphi_2 \times rand \times (gBest - x_i) \quad (3)$$

where θ is the inertia factor and *rand* is a uniformly distributed random number in [0,1]. Later Clerc[4] introduced a constriction factor K for more efficient control and constraining of velocities. Then Eq.1 was modified as:

$$v_i = K \times \left(v_i + \varphi_1 \times rand \times (pBest_i - x_i) + \varphi_2 \times rand \times (gBest - x_i)\right) \quad (4)$$

Here *K* can be expressed as:

$$K = \frac{2}{\left|2-\varphi-\sqrt{\varphi^2-4\varphi}\right|} \quad (5)$$

where $\varphi = \varphi_1 + \varphi_2$. The value $\varphi > 4$, prevents the explosion of the system [5], which can occur when particle velocities increase without control. According to [2], successes of inertia and constriction factor equations are problem dependent. Therefore, in this paper both equations are used and the best result is taken.

3 Related Work

As mentioned earlier, a lot of effort has been put into improving the quality of the basic PSO. Since the main focus of this work is on utilizing additional memory, we mainly try to mention studies in which additional external and internal memory (repository) have been employed in order to improve the basic PSO.

Memory based PSO proposals are mainly concentrated on various methods, by which the local and global best positions are selected and used. The work of Coello Coello et al. [6] is one of the first in this respect. In this work, the global best is determined by selection of a non-dominated solution from the external memory. Local best is updated with respect to Pareto dominance. Hu et al. [7] extended their work using dynamic neighborhoods and employed an external memory to memorize all potential Pareto optimal solutions. In their work the global best is selected among

the candidates in the external memory by defining a neighborhood objective and an optimization objective. The global best is found among the particle's neighbors, subject to the defined neighborhood objective.

One other attempt to employ additional memory in PSO is the work of Wang [8]. In this method, The Triggered Memory-Based PSO, they try to adjust the parameters of the PSO in dynamic environments where the characteristics of the search space change over time. However, this method is novel and successful in terms of presenting the effects of utilizing additional memory in PSO. In their work a certain, predefined number of globally best positions, are kept and re-injected into the search population when necessary. This method is particularly successful when the location of the optima change over time.

One other successful work which utilizes external memory is the work of Acan [9]. In their work a single globally best position is kept along with a certain number of globally worst positions. A crossover operator is used to replace a randomly chosen set of particles after each iteration.

Yet in another work, Acan [2] introduced a hybrid method where there is a global memory and a local memory for each particle. A colony, consisting of the local and global positions is then constructed and at each evaluation, members of the colony are used to update the velocity and position of the particle in process. Then the new positions are evaluated where the best replaces the particle's current velocity.

There are some other approaches, which employ additional memory and/or hybridization or other techniques. Additional information can be found in [2,10]. The main idea in almost all of these memory utilizing approaches is to re-inject the globally best position into the population during the search. In our study, however, the main idea is to deduce information from the contents of the external memory in order to affect the velocity of the particles towards the global optima.

4 The Proposed Approach

Based on observations of the application of evolutionary algorithms on robot motion, one can say that sometimes, parts of the new individuals are close to those of the optimal solution. Only a few parameters in the current individual are different and are far from being a part of the optimum solution. Although the number of these parameters may be small, their effect may cause the performance of the individual to be closer to that of the worst case.

Considering a landscape in which we have many local minima and maxima points, one can also say that, while updating the position of the individual, keeping it away from the maxima (in a minimization problem) will increase the speed of convergence to the minima. But this alone might not be sufficient, since there are problems for which both minima and maxima are in close neighborhoods or when there are many local minima. Then we need to equip the optimization algorithm with the ability to escape from the maxima and move towards the minima even when the minima are close to both the current position of the particle and the maxima.

In this work, we propose an algorithm, in which the particles are guided away from the closest worst location by correcting their positions to the location which they are supposed to be, prior to updating their positions. In order to do this, we construct two

lists for the so far found global best and global worst positions. Each of the two lists is in fact, separate external memories of the PSO. Before updating the velocity of the particles, we scan the external memory that keeps the best positions and determine the best location which is closest to the particle. We call this CB. Then we scan the external memory that keeps the global worst positions and in a similar way, we find the global worst location closest to the particle. We call this position CW. In our experiments Euclidean distances are used. However, any other distance measure may also be used for this purpose.

After choosing the closest best and worst elements from the external memories with respect to the particle, we measure the similarity of the particle to each of these elements. Eqs.7 and 8 calculate the similarity between the closest best and the particle (C_1) and the similarity between the closest worst and the particle (C_2) respectively.

$$c_1 = \frac{\sqrt{\sum_{i=1}^{D}(CB - x_i)^2}}{(x_u - x_l)} \tag{7}$$

$$c_2 = \frac{\sqrt{\sum_{i=1}^{D}(CW - x_i)^2}}{(x_u - x_l)} \tag{8}$$

Here $x_u$and x_l are the upper and lower bound values for each dimension. In our experiments this value range has been considered to be equal for all dimensions. The position correction coefficient (C) can be defined as in Eq.9.

$$C = \frac{|c_1 - c_2|}{(c_1 + c_2)} \tag{9}$$

We are now able to present our algorithm as follows. First, we define two external memories: one contains a fixed number of global best positions and the other contains a fixed number of global worst positions. The sizes of these two external memories are not necessarily equal. To initiate both memories, we first run the basic PSO until the iteration count is equal to the maximum size of the two external memories. At each iteration, we insert the global best and worst positions into the corresponding external memories.

After initialization, we can calculate the C coefficient in Eq. 9. In other words, after each evaluation of all the particles, we refresh the external memory contents and based on the information available in the external memories we calculate the coefficient. Then according to a particular probability, we either use the basic equation or the following equations which are modified forms of Eqs. 3 and 4 respectively:

$$v_i = \omega.(v_i + C) + \varphi_1.rand.(lBest - x_i) + \varphi_2.rand.(gBest - x_i) \tag{10}$$

$$v_i = K.\big((v_i + C) + \varphi_1.rand.(lBest - x_i) + \varphi_2.rand.(gBest - x_i)\big) \tag{11}$$

In Eqs. 10 and 11, we add the coefficient C to the current velocity of the particle, in order to move the particle away from the worst position towards the closest best. When we use Eq.10 for velocity update, based on observations, we can say that

decreasing the inertia linearly, will decrease the effect of the coefficient and yield early convergence.

The pseudo code of the algorithm is shown in Fig. 1. In the algorithm, ω is the inertia which is only applied to Eqs. 10 and 11. Eqs. 3 and 4 use a fixed inertia value of 0.99. $\Delta\omega$ is the inertia decreasing factor.

```
1-  Algorithm CBCW_PSO
2-  begin
3-  Randomly initialize the particles within their ranges and set all velocities to zero
4 – Define external memory for best positions (bests) . Initialize to all zeros
5 - Define external memory for worst positions (worsts). Initialize to all zeros
6 - Until a certain number of iterations is reached or convergence do
7 -   for each particle x_i do
8 -   x_i = x_i + v_i
9 -   if particle position at each dimension exceeds its allowed range
10-       x_i = random from range
11-   val = evaluate particle
12-   update the local best
13-   replace the worst in bests array with gbest
14-   replace the best in worsts array with gworst
15-   if iteration count > max(size(bests), size(worsts))
16-     for each particle x_i
17-       CB = closest best to particle
18-       CW = closest worst to particle
19-       Calculate c_1 and c_2 according to Eqs. 7 and 8
20-       calculate C according to Eq. 9
21-       for each dimension
22-           with probability p
23-           update the position like the basic PSO in Eqs. 3 and 4
24-           otherwise
25-           update the position using Eqs. 10 and 11
26-       if ω ≥ lower bound + ε
27-           ω = ω − Δω − ε          // decrease the inertia
28-       else
29-           ω = upper bound – ε    // resetting the inertia
30- end
```

Fig 1. The pseudo code for CBCW-PSO algorithm

Since this algorithm is based on the distance of the particle to its globally closest best and globally closest worst, we refer to it as CBCW-PSO from here on. Decreasing the inertia (ω in Fig. 1) means that during this period we increase the importance of the social and cognitive factors and pay less attention to the actual velocity of the particle. In cases where the particle is stuck in local optima, using a linearly decreasing inertia leaves the particle with a velocity less than what the particle requires to escape those optima. This is why we decrease the inertia value linearly until a certain threshold is reached. At such a point, the inertia is reset to its upper bound. Here we defined a lower and upper bound value for the inertia. In order

to achieve diversification, some randomness is added to the upper and lower bounds and the decreasing factor ($\Delta\omega$ in Fig. 1).

5 Experiments

The performance of the proposed algorithm is compared with the performance of the improved PSO presented in [2] called CLPSO. This work has been chosen due to its success over the performance of the basic PSO and the extent of the range of problems with which it has been tested. Since an extensive comparison between PSO and CLPSO has been presented in [2], we do not compare our proposed algorithm with PSO. To see the comparison results between the CLPSO and the basic PSO, please refer to [2].

Several unimodal and multimodal benchmark problems have been adopted from [11] and [12]. The list of the test functions and some of their characteristics can be seen in Table 1. In the table "Min" column gives the optimum value of the function, the "Range" column gives the defined range of the parameters. The range is same for all dimensions. Functions $f_1 - f_8$ are unimodal. Function $f_1$is the shifted sphere function. Function f_2 produces hyper ellipsoids; it is continuous and convex. The f_3 function is another simple unimodal function. Function f_4 consists of plateaus. Function f_5 is referred to as sum of different power functions. Function f_6 is a noisy problem. Function f_7 is the high conditioned elliptic function. Function f_8 is the Rosenbrock function. The global optimum lies inside a long narrow parabolic shaped valley. Finding the valley itself is trivial. However, finding the global optima is considered to be a difficult task. Function f_9 is the Schwefel function. This function is deceptive, in that the global minima are geometrically distant over the parameter space from the next best local minima. Therefore the search algorithms are potentially prone to converge in the wrong direction. Function f_{10} or the Rastrigin function, is based on the function of De Jong with the addition of cosine modulation in order to produce frequent local optima. This function is highly multimodal, however the location of local minima are regularly distributed. Function $f_{11}$is the Griewank function and has similar properties to the Rastrigin function. f_{12} is the Ackley function and is multimodal. $f_{13}$is the Weierstrass function. This function is a difficult, multimodal function with multiple local optima. Functions $f_{14} - f_{16}$ are easy to solve, two dimensional functions.

5.1 Experimental Design

The algorithm settings for which the experiments have been done are as follows:
$\Delta\omega = 10^{-5}$, $\varepsilon = 10^{-6}$, $lower\ bound = 0.95\ (0.8\ for\ f_9)$, $upper\ bound = 0.99$. The upper and lower bound of ω as well as $\Delta\omega$ are randomized with a small number, ε. The value of ε has been chosen to be smaller than the value of $\Delta\omega$ to prevent ω from exceeding its boundaries. Choosing a small value for ε also makes sure that ω's value maintains a smooth and slow motion between its boundaries as if $\Delta\omega$ where not

randomized. The value for $\Delta\omega$ is chosen such that it shows a decrement rate close to that of [2] for sake of comparison. The upper bound value is used as suggested in several publications and its validity has been verified in our experiments. The lower bound, however, is experimentally tuned.

Table 1. Different benchmark functions with which the proposed algorithm has been tested.

Benchmark Problems	Min	Range
$f_1(\vec{x}) = \sum_{i=1}^{D} x_i^2$	0	$-5.12 \le x_i \le 5.12$
$f_2(\vec{x}) = \sum_{i=1}^{D} \sum_{j=1}^{i} x_i^2$	0	$-65 \le x_i \le 65$
$f_3(\vec{x}) = \sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} x_i$	0	$-10 \le x_i \le 10$
$f_4(\vec{x}) = \sum_{i=1}^{D} \left(\left\lfloor x_i + \frac{1}{2} \right\rfloor\right)$	0	$-100 \le x_i \le 100$
$f_5(\vec{x}) = \sum_{i=1}^{D} \lvert x_i^{i+1} \rvert$	0	$-1 \le x_i \le 1$
$f_6(\vec{x}) = \left(\sum_{i=1}^{D} (i+1) x_i^4\right) + rand[0,1[$	0	$-5.12 \le x_i \le 5.12$
$f_7(\vec{x}) = \sum_{i=1}^{D} (10^6)^{\frac{i-1}{D-1}} x_i^2$	0	$-100 \le x_i \le 100$
$f_8(\vec{x}) = \sum_{i=1}^{D} (100 \times (x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$	0	$-30 \le x_i \le 30$
$f_9(\vec{x}) = \sum_{i=1}^{D} -x_i \times \sin(\sqrt{\lvert x_i \rvert})$	-12569.5	$-500 \le x_i \le 500$
$f_{10}(\vec{x}) = \sum_{i=1}^{D} (x_i^2 - 10 \times \cos(2\pi x_i) + 10)$	0	$-5.12 \le x_i \le 5.12$
$f_{11}(\vec{x}) = \frac{1}{4000}\left(\sum_{i=1}^{D} x_i^2\right) + \left(\prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i+1}}\right)\right) + 1$	0	$-600 \le x_i \le 600$
$f_{12}(\vec{x}) = -20 \times \exp\left(-0.2 \times \sqrt{\frac{1}{n}\sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{n} cos(2\pi x_i)\right) + 20 + e$	0	$-32 \le x_i \le 32$
$f_{13}(\vec{x}) = \sum_{i=1}^{D} \left(\sum_{k=0}^{Kmax} a^k \cos\left(2\pi b^k (x_i + 0.5)\right)\right) - D \sum_{k=0}^{Kmax} (a^k \cos(2\pi b^k \times 0.5))$	0	$-0.5 \le x_i \le 0.5$
$f_{14}(\vec{x}) = 4x_0^2 - 2.1x_0^4 + \frac{1}{3}x_0^6 + x_0 x_1 - 4x_1^2 + 4x_1^4$	1.0316	$-5 \le x_i \le 5$
$f_{15}(\vec{x}) = \left(x_1 - \frac{5.1}{4\pi^2}x_0^2 + \frac{5}{\pi}x_0 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_0) + 10$	0.398	$-5 \le x_i \le 15$
$f_{16}(\vec{x}) = \frac{-(1 + \cos(12\sqrt{x_0^2 + x_1^2}))}{\frac{1}{2}(x_0^2 + x_1^2) + 2}$	0	$-5.12 \le x_i \le 5.12$

The value of the probability p, which determines whether to use Eqs. 3 and 4 or Eqs. 10 and 11, has a direct effect on the convergence speed of the algorithm. For lower values, the algorithm switches to Eqs. 10 and 11. This adds an extra amount to the velocity of the particle in the direction of its closest best position and opposite the direction of its closest worst position. However, there is a trade-off. Frequently using the extra velocity for a long time, results in convergence in a wrong direction. This is due to the fact that local optima may be scattered and far from each other but at a relatively equal distance with respect to the particle. This is why a value above 0.5 has been considered for the probability p (0.8 in general and 0.5 for the f_9 problem). Our experiments show that setting the probability p proportional to $(1 - \omega)$, increases the convergence speed, since binding p to the value of ω, which is a randomized value,

provides more diversification. Replacing the linearly decreasing inertia in the original form of Eq. 10 with an inertia value, which is subject to multiple restarts, also shows to be a very influencing factor on convergence speed. This is the methodology of choice in our experiments.

The maximum number of swarm evaluations is set to 40000 for the CBCW-PSO and equivalently to 2000 for the CLPSO. CLPSO runs an internal loop for $m \times n$ number of elements inside the colony. Here m represents the local memory size and n represents the global memory size. Referring to the work of Acan [2], the recommended local and global memory sizes are 5 and 4 respectively. Then setting the maximum evaluation to 2000 does in fact perform a total of 40000 swarm evaluations which is equal to that of CBCW-PSO. The size of the external memory for best and worst global positions is 2 and 4 respectively. The swarm size in all the experiments is constant and set to be equal to 30, which is the dimension of the functions. All the experiments are performed over 10 runs for each problem and algorithm. The test platform is an 8 core (3.2 GHz) system running on Ubuntu 10.04 with 4 GB RAM. The test results can be seen in Table 2. Outcomes less than or equal to 10^{-5}, is considered as zero.

Table 2. Result of applying both CLPSO and CBCW-PSO on various benchmark functions. The bold numbers, represent the best results.

Functions	**Dim**	**CLPSO**			**CBCW-PSO**			**t-test**
		Min	**Max**	**Avg**	**Min**	**Max**	**Avg**	
$f_1(\vec{x})$	30	0	0	0	0	0	0	S
$f_2(\vec{x})$	30	0	0	0	0	0	0	S
$f_3(\vec{x})$	30	0	0	0	0	0	0	S
$f_4(\vec{x})$	30	0	0	0	0	0	0	S
$f_5(\vec{x})$	30	0	0	0	0	0	0	S
$f_6(\vec{x})$	30	0	0	0	0	0	0	S
$f_7(\vec{x})$	30	0	0	0	0	0	0	S
$f_8(\vec{x})$	30	0.033	7.93	3.899	**0.001**	**4.001**	**0.911**	**S+**
$f_9(\vec{x})$	30	-12451	-11622	-12107.5	**-12569.5**	**-12332**	**-12438**	**S+**
$f_{10}(\vec{x})$	30	48.915	237.94	173.8	**0**	**1.989**	**0.994**	**S+**
$f_{11}(\vec{x})$	30	0	0.11	0.042	**0**	**0.041**	**0.012**	**S+**
$f_{12}(\vec{x})$	30	2.2209	5.22	3.318	**0**	**0**	**0**	**S+**
$f_{13}(\vec{x})$	30	2.7555	7.19	5.922	**0**	**1.57**	**0.15**	**S+**
$f_{14}(\vec{x})$	2	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316	-1.0316	S
$f_{15}(\vec{x})$	2	0.398	0.398	0.398	0.398	0.398	0.398	S
$f_{16}(\vec{x})$	2	0	0	0	0	0	0	S

5.2 Results

In Table 2, the minimum, maximum and average values obtained over 10 runs are given. The “dim” column shows the number of dimensions used for each function. Better results are shown in bold in the table. A t-test at a significance level of 0.95 has been performed to test for statistical significance of the differences. In the last

column, S+ means that CBCW-PSO results are statistically significantly better than those of CLPSO and S means no statistically significant differences were found.

As it is shown in Table 2, the CBCW-PSO, outperforms the CLPSO in functions$f_8 - f_{13}$. These functions are all multimodal functions for which CLPSO and PSO are unable to converge to the optima in the experiments. The performance of the CBCW-PSO is better also in terms of the maximum value and the average fitness.

As mentioned earlier, the real purpose and motivation behind the design of CBCW-PSO is to achieve acceptable results in lower iteration counts. In other words, there is a need for an optimization algorithm as good as PSO or improved versions of it, like CLPSO, which yields at least the same results in a fewer number of evaluations. We have shown that CBCW-PSO achieves better results, compared to CLPSO in a fixed number of iterations. Although this alone indicates a better convergence speed, still presenting this superiority in an analytical form is needed. There are functions in Table 1 for which both algorithms find the global optima. In fact, looking at Table 2, one can see that in functions $f_8 - f_{13}$, CLPSO is not able to find the optima, given a fixed limit on maximum evaluation counts. Again Table 2 shows that CBCW-PSO is able to find the optima in almost all of the benchmark functions (in f_8, although both algorithms are unable to find the optima, CBCW-PSO still gives a closer value to the optimum). The question is how much faster CBCW-PSO is able to find the optima.

Table 3 shows the average number of evaluations until the first hit, where the global best in the algorithm is the same as the location of the optima, if found. The last column of this table represents the 95% significance t-test results with first hit time as the test parameter.

Table 3.Comparing CBCW-PSO and CLPSO in terms of average number of evaluations until the first hit for each of the benchmark functions for which at least one of the two algorithms finds the optima. For f_8 none of the methods located the optimum.

Functions	**CLPSO**	**CBCW-PSO**	**t-test**
$f_1(\vec{x})$	122820	**15255**	**S+**
$f_2(\vec{x})$	165900	**31419**	**S+**
$f_3(\vec{x})$	299880	**122265**	**S+**
$f_4(\vec{x})$	1199400	**77832**	**S+**
$f_5(\vec{x})$	100920	**2340**	**S+**
$f_6(\vec{x})$	122280	**7605**	**S+**
$f_7(\vec{x})$	247860	**33582**	**S+**
$f_8(\vec{x})$	-	-	
$f_9(\vec{x})$	-	**1109850**	**S+**
$f_{10}(\vec{x})$	-	**1076010**	**S+**
$f_{11}(\vec{x})$	1094340	**963669**	**S**
$f_{12}(\vec{x})$	-	**57153**	**S+**
$f_{13}(\vec{x})$	-	**646980**	**S+**
$f_{14}(\vec{x})$	3180	**531**	**S+**
$f_{15}(\vec{x})$	2700	**489**	**S+**
$f_{16}(\vec{x})$	2120	**775**	**S+**

Also, convergence plots of four functions, namely f_8, f_9, f_{12} and f_{13}, given as best fitness plots, averaged over 10 runs has been shown in Fig 2. Functions f_9, f_{12}, f_{13} show how CBCW-PSO performs better than CLPSO since in all three cases CLPSO is unable to converge to the optimum while CBCW-PSO converges precisely. Function f_8 is chosen since it demonstrates the fact that although both algorithms fail to find the optimum, CBCW-PSO maintains a better converging behavior.

As can be seen (both in Table 3 and Fig. 2), CBCW-PSO, needs a shorter number of evaluations to find the optima. One interesting point is that, although CLPSO seems to converge faster than CBCW-PSO in case of function f_9, due to its random choice of best position in local memory, it jumps to some other local optima in later evaluations. This shows that CBCW-PSO maintains a monotonically decreasing behavior with respect to the best fitness.

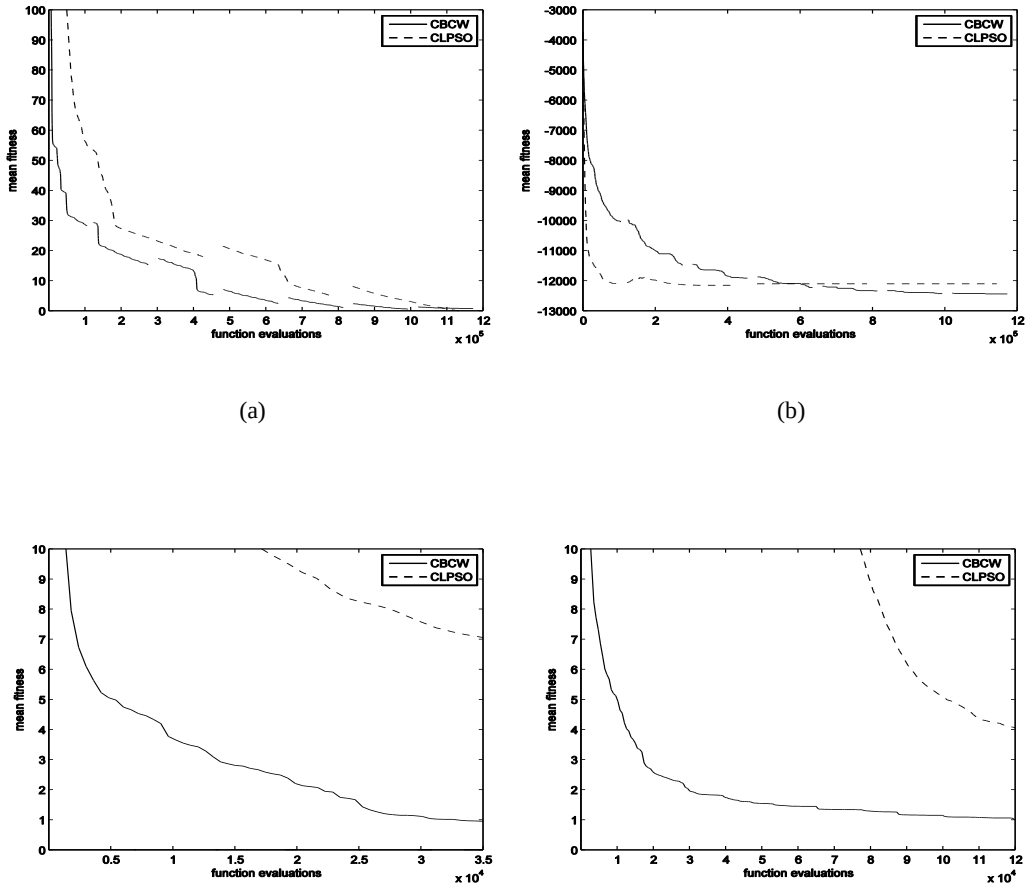

Fig 2.The evolution of the best fitness averaged over ten runs for each function. (a) f_8 (b) f_9 (c) f_{12} and (d) f_{13}

6 Conclusions and Future Work

In this paper, we proposed a new and novel PSO, based on external memory for both global best and worst positions. We showed that the proposed algorithm outperforms a recent improvement of PSO, CLPSO, in almost all of the tested benchmark functions, in terms of precision and convergence speed.

However, the algorithm settings may be problem dependent. In case of function f_9, the value chosen for some of the parameters are different than the general settings. Although applying the general settings of the algorithm in this case (function f_9), also yields good results, there exists settings which yield better results. Based on this, we will be introducing adaptive parameter settings for the algorithm in our future studies. Also, after improving our algorithm, we will use it in a realistic robot motion simulator environment.

References

[1] J. Kennedy, R. C. Eberhart, "A New Optimizer Using Particle Swarm Theory", Sixth International symposium on Micro Machine and Human Science, IEEE, 1995

[2] A. Acan, "A Memory-Based Colonization Scheme for Particle Swarm Optimization", IEEE Congress on Evolutionary Computation (CEC) , 2009.

[3] Y. Shi, R. C. Eberhart, "Parameter Selection in Particle Swarm Optimization", 7th International Conference on Evolutionary Programming VII. (EP), 1998

[4] M. Clerc, "The Swarm and The Queen: Towards a deterministic and Adaptive Particle Swarm Optimization", IEEE Congress on Evol. Comp. (CEC), 1999.

[5] M. Clerc, J. Kennedy, "The Particle Swarm – Explosion, Stability, and Convergence in a Multidimensional Complex Space", IEEE Transactions on Evol. Comput., vol. 6, no. 1, pp. 58–73, Feb. 2002.

[6] C. A. Coello Coello, M. S. Lechuga,"MOPSO: A Proposal for Multiple Objective Particle Swarm Optimization", IEEE Congress on Evol. Comp. (CEC), 2002.

[7] X. Hu, R. C. Eberhart, Y. Shi,"Particle Swarm with Extended Memory for Multiobjective Optimization", Proceedings of the IEEE Swarm Intelligence Symposium (SIS), 2003.

[8] H. Wang, D. Wang, S. Yang, "Triggered Memory-Based Swarm Optimization in Dynamic Environments", Applications of Evolutionary Computing, 2007

[9] A. Acan , A. Gunay, "Enhanced Particle Swarm Optimization Through External Memory Support", IEEE Congress on Evolutionary Computation (CEC), 2005.

[10] X. Hu, Y. Shi, R. C. Eberhart, "Recent Advances in Particle Swarm", IEEE Congress on Evolutionary Computation (CEC), 2004.

[11] R.Thomsen, J. Vesterstrom, "A Comparative Study of Differential Evolution, Particle Swarm Optimization, and Evolutionary Algorithms on Numerical Benchmark Problems", IEEE Congress on Evolutionary Comp. (CEC), 2004.

[12] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y. -P. Chen, A. Auger, S. Tiwari, "Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization", IEEE Congress on Evol. Comp. (CEC), 2005.