PSO under an Adaptive Scheme

Mihaela Breabăn
Faculty of Computer Science
Alexandru Ioan Cuza University, Iasi, Romania
pmihaela@infoiasi.ro

Henri Luchian
Faculty of Computer Science
Alexandru Ioan Cuza University, Iasi, Romania
hluchian@infoiasi.ro

Abstract – This paper presents an attempt to transform PSO into a self-adaptive algorithm based on specific swarm-inspired operators. New features are introduced: spatial expansion intended to overcome premature convergence (an algorithm called Improved PSO, IPSO) and auto-adaptation (an algorithm called Adaptive PSO, APSO). Experiments show that APSO and IPSO outperform the basic PSO on benchmark problems, proving their efficiency especially on multimodal functions.

1 INTRODUCTION

The PSO model was introduced in 1995 by J. Kennedy and R.C. Eberhart, being discovered through simulation of a simplified social model such as fish schooling or bird flocking [1]. It was originally conceived as "a method for optimization of continuous nonlinear functions". Latter studies showed that PSO can be successfully adapted to solve integer problems [2] and even combinatorial problems [3, 4, 5, 6].

Like Ant Colony, PSO is a swarm intelligence technique and shares similarities with evolutionary computation techniques such as Genetic Algorithms (GA). They all are population-based heuristics and work in the same way: they update the population by applying some operators according to the fitness information, in order to reach better solution areas.

A major problem of all these algorithms and particularly of PSO in multi-modal optimization is premature convergence, which results in sub-optimal solutions [7, 8, 9]. This problem has been addressed in several papers and solutions include: addition of a queen particle [10], alternation of the neighborhood topology [11], introduction of subpopulations [12], giving the particles a physical extension [13], alternation between phases of attraction and repulsion [14], giving different temporary search goals to groups of particles [15], and giving particles quantum behavior [16].

Another crucial problem is parameter control. The values and choices for some of these parameters may have significant impact on the efficiency and reliability of the PSO. There are several papers that address this problem; in most of them, values for parameters are established through repeated experiments [17, 18, 19] but there also exist attempts to adjust them dynamically, using evolutionary computing [20].

This paper addresses both problems: parameter control by adjusting the parameters in an adaptive way and premature convergence by introducing specific operators, which allow for a better exploration of the search space and for escaping local minima.

2 BASIC PSO

PSO consists of a group (swarm) of individuals (particles) moving in the search space, their trajectory being determined by the fitness values found so far. Each particle is represented by a vector x of length n indicating the position in the n-dimensional search space and has a velocity vector v used to update the current position. The velocity vector is computed following the rules:

1) every particle tends to keep its current direction (an inertia term);

2) every particle is attracted to the best position p it has achieved so far (a memory term);

3) every particle is attracted to the best particle g in population (the particle having the best fitness value); there are versions of the algorithm in which the best particle g is chosen from topological neighborhood.

Thus, the velocity vector is computed as a weighted sum of the three terms above. The formulas used to update each of the individuals in the population at iteration $t+1$ are:

$$v[t+1] = w_1 v[t] + w_2 rand() (p-x) + w_3 rand ()(g-x) \quad \textbf{(1)}$$

$$x[t+1] = x[t] + v[t+1] \quad \textbf{(2)}$$

The role played by the inertia weight was compared to that of the temperature parameter in Simulated Annealing [9]. A large inertia weight facilitates a global search while a small inertia weight facilitates a local search.

The parameters w_2 and w_3 are called generically learning factors; because of their distinct roles, w_2 was named the cognitive parameter (it gives the magnitude of the information gathered by each individual) and w_3 the social parameter (it weights the cooperation between particles).

Another parameter used in PSO is the *maximum velocity* which determines the maximum change each particle can take during one iteration. This parameter is usually proportional with the search domain.

3 An adaptive approach

3.1 Adaptive parameter control

One can think of the terms which compute the velocity vector as of specific operators which can be applied one

by one, having different goals and different weights at each moment. Since the adaptive way of modifying the parameters proved to be efficient for algorithms such as GAs, we adopted the same strategy for PSO. At first, we studied the impact this dynamic parameter control has on basic PSO after which we tried to introduce some new operators in order to better explore the search space.

The pseudo-code for our algorithm (Adaptive PSO) is given in *Figure1*.

```
begin
initializePopulation();
initializeVelocities();
initializePrameters();
assignFitness();
while not stopCondition do
    for each operator
        applyOperator(population, operator, weight)
        assignFitness()
        modify(weight);
end

procedure applyOperator(population, operator, weight)
    for each individual in population
         operator (individual, weight);
```

Figure1: The Adaptive PSO algorithm

Each of the three terms used to compute the velocity vector in basic PSO has a specific goal along the search: the inertia term is necessary for a better exploration of the search space, the memory term is used for locating new promising areas and finally, the third term exploits the search regions around the best individual in population. From this point of view we can consider them distinct operators which can be applied independently while we trace the contribution that each of them has in improving the quality of solutions.

As Eberhart and Shi mentioned in [9], PSO is the only evolutionary algorithm that does not implement survival of the fittest. In our algorithm we keep out this strategy too but we do use some kind of elitist approach. We do not use the elitism with its usual meaning: we do not include the best solution found so far in the next generation by replacing the worst individual but we do store it and use it for computing the third term in (1) – move towards the best solution.

Applying the Adaptive PSO scheme on the basic PSO we compute the total velocity in three steps, and at each step we update each individual in the current population. Thus, an operator from the algorithm scheme shown in *Figure1* applied to an individual can be viewed as the computation of some velocity and the update of an individual's position:

```
procedure operator (individual, weight)
    someVelocity = weight * computeVelocity(individual);
    individual.position= individual.position+someVelocity;
```

Figure2: Operator scheme

The difference between operators is given by *computeVelocity* function from *Figure2* which is specific to each operator and is implemented to achieve a specific goal. For the first operator (the inertia operator) *computeVelocity* function consists in using the total velocity computed in the previous iteration. The second operator computes the velocity from the individual's current position and the best previous position it ever encountered, while the third operator uses the best position encountered so far by any individual.

So far, our algorithm is just a rephrasing of the basic PSO. The importance of this new formulation is that it gives us the possibility to judge and establish the efficiency of each operator at different moments along the search process.

Each operator is applied with a certain weight (we can consider the weights w_1, w_2 and w_3 in *formula 1*), controlling in this way its influence. Based on the history of the search we increase or decrease these weights according to the success record of operators in locating better solutions. The simplest way to modify the weight of an operator (which is the one we use) is to compare the best individuals in population before and after the operator was applied on the entire population. If the use of the operator produces a better solution than the best individual in the previous generation, the magnitude of this operator is increased; otherwise, its magnitude is decreased.

In more detail, at every iteration the three operators are applied in turn, in other words the position of each individual is updated three times per iteration using each of the three terms in *formula1*. After an operator is applied to all individuals in the population, each particle is evaluated. If the best individual in current population has a better fitness value than the best individual in the previous population (before applying the operator), the corresponding weight is increased. In this way each of the three weights can be modified at every iteration. The order in which we apply the operators at each iteration is the one from *formula 1* but it does not influence the performance of the algorithm.

It is obvious here the importance of using a non - elitist approach; however, the use of the best-so-far solution encountered along the search process instead of the best current solution for the exploitation operator gave far better results.

3.2 More operators

Inspired from the way birds fly, other features can be noticed in nature and translated into operators for PSO. Avoiding collisions, expanding, grouping together or flying in a specific direction can be simulated to improve the search process.

Using the adaptive scheme described above, we tried to add some new operators; however, the use of one or more extra operators slowed down the movement towards the global optimum. One reason for this efficiency loss might be the opposite effects of new operators: when applied on the same individual with approximately equal weights, they nullify each other. In the rest of this paper we will refer to this approach which uses new operators as *Improved PSO*.

Since quasy-equal weights slow down the Improved PSO and this may happen under the adaptive scheme described in section 3.1, we adopted a modified approach: at any given moment one operator should have a larger

magnitude than the others have. When and how this trade-off should be made is decided as well in an adaptive way.

This approach differs from the basic PSO not only in the way the parameters are set but also in the formula used to compute the velocity. In our first approach we computed the velocity exactly as in the basic PSO, the only change being that the terms in the formula (1) are computed one at a time. The modified approach does not use the same formula: new terms are introduced and some existing ones are removed. Four distinct operators control the swarm in the following way:

- one operator shifts the entire swarm in a given direction
- an expansion operator increases the swarm dispersion
- a grouping operator contracts the swarm
- the last operator performs a local search.

The goal of the first two operators is to perform a good exploration of the search space while the latter two exploit the promising areas. As in the basic PSO, these operators consist in computing some velocities and updating the individuals' positions. The first operator is the simplest one: a velocity vector is initialized randomly and then used to update each individual in the swarm. The effect is a spatial position change of the entire swarm as a whole. The expansion operator is a bit more complicated: it uses the neighbors of each particle and works like a repulsion force. In addition, each particle is pushed away by the best particle found so far. The grouping operator is identical to the third term in the basic PSO: the particles are attracted by the best individual found so far. The last operator was conceived at first as a local search performed independently by each individual along a predefined number of iterations. Then we replaced it by the second operator from the basic PSO which seems to give comparable results: each particle is attracted towards its best previous position.

Repulsion schemes have been previously used in order to overcome premature convergence. In [14] phases of attraction alternate with phases of repulsion: the terms in *formula 1* are used to compute the velocity with different signs for the two different phases. In [21] particles are repelled away from the detected local minimizers.

In IPSO another repulsion scheme is used. For the expansion operator, two velocity vectors are computed: one is given by the repulsion from the neighbors (we call it *cohesionRepulsion*) and the other by the repulsion from the best individual in population (*globalRepulsion*):

cohesionRepulsion = -(n - x)
globalRepulsion = -(g-x)

where n is the gravity center of all neighbors, g is the position of the best particle and x is the current position of the particle for which the velocity vector is computed.

For the grouping operator, the vector *globalAttraction* is computed:

globalAtraction = (g-x) = - globalRepulsion

For the last operator the memory term from the basic PSO is calculated:

memory = (p - x)

The total velocity with weights, computed at each iteration, is given by:

*v = WEIGHT_SHIFT * randomVelocity +*
WEIGHT_COHESION (n - x) +
WEIGHT_GLOBAL (g - x) +
WEIGHT_MEMORY (p – x)

This formula and the basic PSO are similar: the only difference seems to be the replacement of the inertia term by two terms - the shifting and the cohesion velocity. The main idea in this approach is to combine the weight values in such a way that, at any given moment, the overall effect is similar to that of one operator. For example, negative weights for the second and the third term and 0 values for the others produce the expansion operator. Thus, unlike in our adaptive PSO from section 3.1, the formula above is computed in one step, changing only the weights as soon as a change of operator is decided. There are no sudden changes: the respective weights are increased/decreased slowly in a number of iterations until the desired effect is accomplished. This is the way the transition from one operator to another is made. One still has to decide when to switch to another operator and to which one exactly. The decision is taken in a simple way: one operator is used as long as it gives good results; to be more specific, if during k iterations an operator does not improve the best solution found so far, it is changed. The operator to be applied next is chosen at random.

4 EXPERIMENTS

4.1 Experiment Design

The three algorithms we tested experimentally were: Basic PSO, Adaptive PSO and Improved PSO.

We tested the algorithms for four functions widely used as benchmark functions for optimization strategies:

Sphere: $f(x) = \sum_{i=1}^{n} x_i^2$

Griewank: $f(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1$

Rosenbrock: $f(x) = \sum_{i=1}^{n-1} 100\,(x_{i+1} - x_i^2)^2 + (x_i - 1)^2$

Rastrigin: $f(x) = \sum_{i=1}^{n} x_i^2 - 10 \cdot \cos(2\pi x_i) + 10$

Asymmetric initialization was used in all cases in order to exclude the optimum from the initialization region [8]. The domain of the functions and the initialization range are shown in *Table 1*.

Function name	Function domain	Initialization range
Sphere	$(-100, 100)^n$	$(50,100)^n$
Griewank	$(-600, 600)^n$	$(300, 600)^n$
Rosenbrock	$(-100, 100)^n$	$(15, 30)^n$
Rastrigin	$(-10, 10)^n$	$(2.56, 5.12)^n$

Table 1: Functions and initialization domains

The common parameters used for the three algorithms we compare are:

- population size: 30
- number of iterations: 10000
- maximum velocity: proportional with the function domain, listed in *Table 2*.

Function name	Max velocity	Neighborhood size
Sphere	100	10
Griewank	100	10
Rosenbrock	50	10
Rastrigin	5	3

Table 2: Maximum velocity and neighborhood size

All experiments were repeated 50 times.

For the basic PSO we used a linearly decreasing inertia weight which starts at 0.9 and ends at 0.2; equal values for the social and cognitive parameters $w_2=w_3=2$ were used.

For the Adaptive PSO all parameters vary between 0.2 and 5 with the increasing/decreasing step 0.3. Each of the three weights is increased or decreased depending on the corresponding operator performance in each iteration: if a better solution than the best individual in the last generation is found, the parameter is increased otherwise it is decreased.

In the Improved PSO the weights are given values between -1 and 1, depending on the operator which has the largest magnitude. At the beginning of each iteration all weights are decreased by multiplying them by 0.95; then the magnitude of the current operator is increased by increasing the absolute value of the corresponding weights. Thus, a new operator gains control while the others fade out. If the new operator in control gives no good results in 5 iterations it gives the control to another randomly chosen operator. The cohesion weight is always maintained below 0, the reason being that keeping a certain level of diversity in the population improves the search results. The neighborhood size is proportional to the function domain and the dimension of the search space (*Table 2*).

4.2 Experimental Results

In order to analyze the performance of our algorithms we made several tests for different dimensions of the search space. *Table 3* shows the average best fitness over 50 runs (10000 iterations/run) obtained for the tested functions in 10, 30 and 50 dimensions. The precision is 10^{-300} so that values below 1E-300 are denoted with 0.

The results show that on all test functions both Adaptive PSO and Improved PSO outperform the basic PSO regarding the convergence time and the quality of solutions; especially in large dimensions the improvements are obvious.

Regarding the number of function evaluations computed by each algorithm, BPSO and IPSO behave similarly: at each iteration only one function evaluation is necessary for each individual. Unlike these two algorithms, APSO makes three function evaluations for each individual per iteration in order to measure the effect of each operator. Since at every iteration at least one function evaluation is computed for each individual, the number of function evaluations highly depends on the size of population (in our experiments set to 30).

The number of function evaluations computed by each algorithm during a run (10000 iterations) is given in *Table4.*

Algorithm	BPSO	APSO	IPSO
No of function evaluations	300000	900000	300000

Table 4: Number of function evaluations in 10000 iterations

Figures 3 to 6 show the performance of the algorithms over the number of function evaluations.

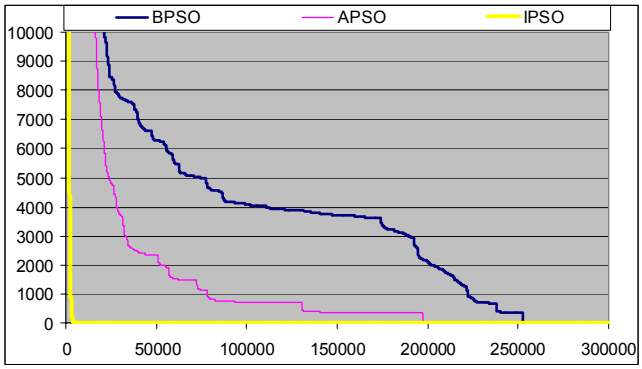

Figure 3: Sphere, 50 dimensions (fitness/function evaluations)

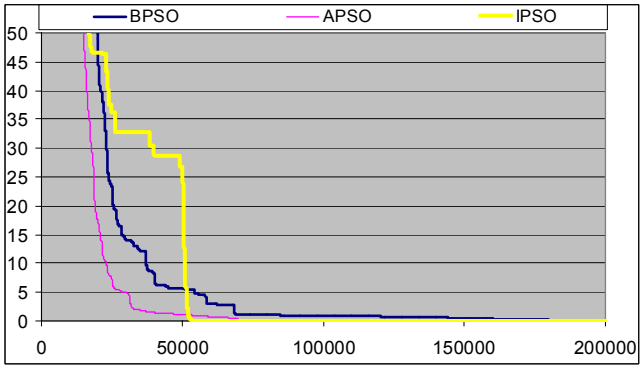

Figure 4: Griewank, 50 dimensions (fitness/function evaluations)

Dimension	10	30	50
Sphere (optimum=0)			
Basic PSO	5.16E-147	5.11E-30	5.88E-11
Adaptive PSO	2.09E-218	2.18E-70	3.09E-29
ImprovedPSO	2.25E-185	1.67E-267	1.5E-268
Griewank (optimum=0)			
Basic PSO	0.067435	0.002483	0.005167
Adaptive PSO	0.066506	0.015658	0.055343
ImprovedPSO	0.073335	0	0
Rosenbrock (optimum=0)			
Basic PSO	1.67816	35.47879	89.62836
Adaptive PSO	2.08E-04	6.52649	43.79519
ImprovedPSO	3.35064	7.48176	11.23911
Rastrigin (optimum=0)			
Basic PSO	3.08207	36.68072	123.40854
Adaptive PSO	0.33165	1.68183	2.12257
ImprovedPSO	1.77787	1.32E-14	5.98E-15

Table 3: Average best fitness values

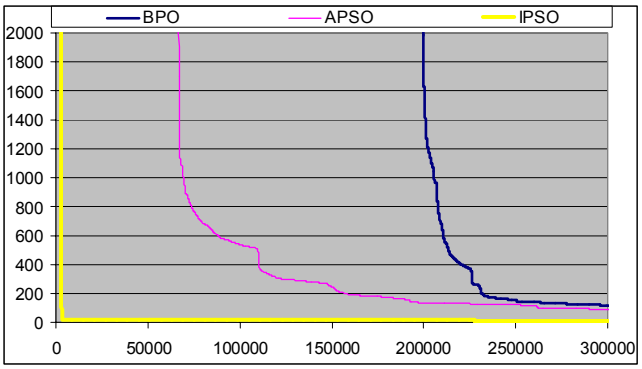

Figure 5: Rosenbrock, 50 dimensions(fitness/function evaluations)

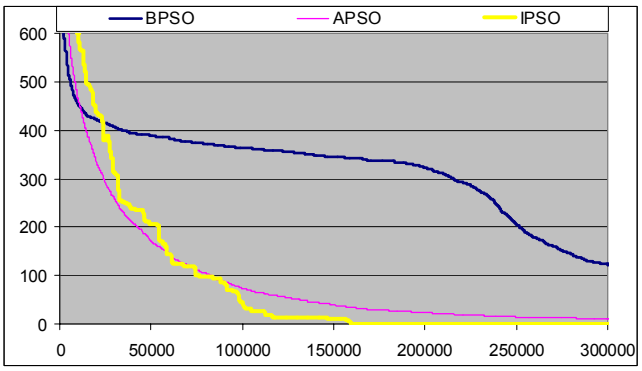

Figure 6: Rastrigin, 50 dimensions (fitness/function evaluations)

An interesting tendency can be observed on the Improved PSO, which gives slightly worse results as the space dimension is decreased on some of the test functions. The reason becomes obvious when some of the particularities of the functions are studied. The Griewank function is highly multi-modal in low dimensions while in higher dimensions it resembles more the plain Sphere function. The Rosenbrock function seems extremely steep when is studied from its domain bounds. The Rastrigin function is highly multimodal in all dimensions.

Adaptive PSO converges much faster than Basic PSO both on uni-modal and highly multi-modal functions. Regarding the quality of solutions, the differences are clear especially for highly multimodal functions in high dimensions.

Improved PSO gives better results than both BPSO and APSO in higher dimensions for all test functions. Although Improved PSO converges a little slower than APSO for highly multimodal functions, it keeps improving the solution even in the latter iterations of the algorithm. For unimodal functions IPSO converges very fast and finds the optimum in less than 1000 iterations.

In order to analyze the contribution of each of the four operators in the search process two runs of IPSO algorithm on Rastrigin function in 50 dimensions are shown in *Figures 7* and *8*. The fitness value is plotted over the iteration number; the colors used indicate the active operator.

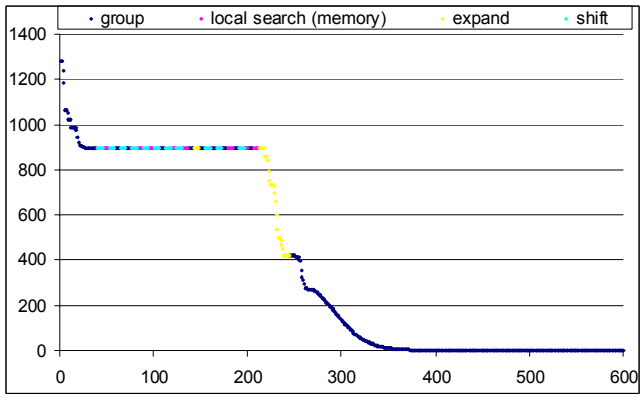

Figure 7: Active operators in IPSO during a run

In both runs the algorithm starts by using the grouping operator which is used as long as it improves the fitness function. When no improvement is made for 5 iterations another operator is randomly chosen and used in the same manner. In this way the control is given dynamically to one of the four operators. In *Figure 7* repeated trials to find an operator which improves the fitness function can be observed during iterations 40-210. Finally, the expansion and the grouping operator take the control for a large number of iterations.

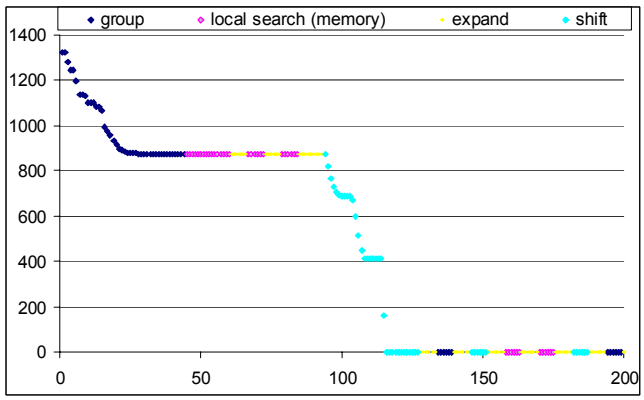

Figure 8: Active operators in IPSO during a run

In *Figure 8* the shift operator improves significantly the fitness function during iterations 92-125 after which other operators try to find better solutions.

The frequency of operators and the moment when a specific operator is used differ for distinct runs of the algorithm; the decisions are taken dynamically and no predefined scheme can be used. No rule regarding the efficiency of the operators or the moment and the order in which they must be applied can be drawn.

5 CONCLUSION

This paper addresses two crucial problems for PSO: parameter control and premature convergence. Two new algorithms are presented: Adaptive PSO (APSO) and Improved PSO (IPSO).

Auto-adaptation is a feature which, included into basic PSO, improves it significantly. Adaptive PSO algorithm

controls dynamically the three parameters in the basic PSO based on the success record of operators.

Other features inspired from nature can be translated into operators for PSO. The Improved PSO algorithm uses new operators intended to overcome premature convergence. An adaptive scheme for parameter control is also used.

Both APSO and IPSO outperform the basic PSO regarding the convergence time and the quality of solutions on benchmark functions.

Bibliography

[1] Kennedy J. and Eberhart R.C. "Particle Swarm Optimization", Proceedings of the 1995 IEEE International Conference on Neural Networks, vol. 4, 1942-1948 IEE Press.

[2] Laskari EC, Parsopoulos KE and Vrahatis MN (2002b) "Particle Swarm Optimization for Integer Programming", IEEE Congress on Evolutionary Computation, pp. 1582–1587

[3] Kennedy J. And Eberhart R. C., "A Discrete Binary Version of the Particle Swarm Optimization", Proc. Of the conference on Systems, Man, and Cybernetics SMC97, pp. 4104-4109, 1997.

[4] Salman A., Ahmad I., and Al-Madani S., "Particle Swarm Optimization for Task Assignment Problem", Microprocessors and Microsystems, 26(2002) 363-371.

[5] Yoshida H., Kawata K., Fukuyama Y., and Nakanishi Y., "A particle swarm optimization for reactive power and voltage control considering voltage security assessment," IEEE Transactions on Power Systems, Vol.15, No.4 pp.1232-1239, 2000.

[6] Wang, K.-P., Huang, L., Zhou, C.-G., and Pang, W. "Particle swarm optimization for traveling salesman problem." Proceedings of International Conference on Machine Learning and Cybernetics 2003, pp. 1583-1585, 2003

[7] Shi Y. and Eberhart R.C. "Empirical Study of Particle Swarm Optimization", Proceedings of the 1999 Congress on Evolutionary Computation, vol3, 1945-1950. IEEE Press

[8] Angeline P.J., "Evolutionary Optimization Versus Particle Swarm Optimization: Philosophy and Performance Differences", Evolutionary Programming VII, (1998) Lecture Notes in Computer Science 1447, 601-610. Springer.

[9] Eberhart R.C. and Shi Y., "Comparison between Genetic Algorithms and Particle Swarm Optimization", Evolutionary Programming VII (1998), Lecture Notes in Computer Science 1447, 611-616. Springer.

[10] Clerc M., "The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization"

[11] Kennedy J., "Small Worlds and Mega-Minds: Effects of Neighborhood Topology on Particle Swarm Performance", Proceedings of the 1999 Congress of Evolutionary Computation, vol. 3, 1931-1938. IEEE Press.

[12] Lovbjerg M., Rasmussen T. K. and Krink T., "Hybrid Particle Swarm Optimizer with Breeding and Subpopulations", Proceedings of the third Genetic and Evolutionary Computation Conference (GECCO 2001).

[13] Krink T., Vesterstrom J., Riget J., "Particle Swarm Optimization with Spatial Particle Extension ", Proceedings of the Congress on Evolutionary Computation 2002.

[14] Riget J., Vesterstrom J, "A Diversity-guided Particle Swarm Optimizer – the ARPSO", EVALife Technical Report no. 2002-02.

[15] Buthainah Al-kazemi and Chilukuri K. Mohan "Multi-Phase Generalization of the Particle Swarm Optimization Algorithm", Proceedings of the Congress on Evolutionary Computation 2002

[16] Jun Sun, Bin Feng, Wenbo Xu, "Particle Swarm Optimization with Particles Having Quantum Behavior", Proceedings of the Congress on Evolutionary Computation 2004, pp. 325-331.

[17] Y. Shi and R. C. Eberhart, "Parameter Selection in Particle Swarm Optimization", Evolutionary Programming VII (1998), Lecture Notes in Computer Science 1447, 591–600. Springer.

[18] Eberhart, R.C., and Shi, Y. (2000), Comparing Inertia Weights and Constriction Factors in Particle Swarm Optimization, 2000 Congress on Evolutionary Computing, vol. 1, pp. 84-88.

[19] Anthony Carlisle, Gerry Dozier, "An Off-The-Shelf PSO",
PSO Workshop, IUPUI, Indianapolis, IN, April, 2001

[20] Gerry Dozier, Gerry Dozier, "New Evolutionary Particle Swarm Algorithm applied to Voltage Control", 14th Power Systems Computation Conference, Sevilla 2002.

[21] Konstantinos E. Parsopoulos and Michael N. Vrahatis, "On the Computation of All Global Minimizers through Particle Swarm Optimization", IEEE Transactions on Evolutionary Computation, pg. 211-224, 2004