Particle Swarm Optimization with Variable Population Size

Laura Lanzarini, Victoria Leza, and Armando De Giusti

III-LIDI (Institute of Research in Computer Science LIDI)
Faculty of Computer Sciences. National University of La Plata
La Plata, Buenos Aires, Argentina

Abstract. At present, the optimization problem resolution is a topic of great interest, which has fostered the development of several computer methods forsolving them.

Particle Swarm Optimization (PSO) is a metaheuristics which has successfully been used in the resolution of a wider range of optimization problems, including neural network training and function minimization. In its original definition, PSO makes use, during the overall adaptive process, of a population made up by a fixed number of solutions.

This paper presents a new extension of PSO, called VarPSO, incorporating the concepts of age and neighborhood to allow varying the size of the population. In this way, the quality of the solution to be obtained will not be affected by the used swarms size.

The method here proposed is applied to the resolution of some complex functions, finding better results than those typically achieved using a fixed size population.

Keywords: Evolutionary Computation, Swarm Intelligence, Particle Swarm Optimization, Function Optimization.

1 Introduction

Optimization, in the sense of finding the best solution or at least an acceptable one for a given problem, is a field of critical importance in the real life. We are constantly solving optimization problems, such as for example the shortest way to get to a place or the organization of our daily duties so as to spend as little time as possible. However, when the problem to solve is extremely complex it is essential to have the computer tools to address it [9].

Due to the great importance of optimization problems, several computer methods have been developed to solve them, which can be generally classified into exact and approximate. At present, in the context of approximate solutions, research has been focused on design and application of metaheuristics, which are based on the integration of local improvement procedures and high level strategies creating efficient processes in terms of computing times and memory space. Such metaheuristics are capable of delivering a good solution, i.e., relatively closed to the optimal, by examining just a small subset of solution

L. Rutkowski et al. (Eds.): ICAISC 2008, LNAI 5097, pp. 438–449, 2008.

of the overall number. Particle Swarm Optimization (PSO) is a metaheuristics which has been successfully used in the resolution of a wider range of optimization problems, including neural network training and function minimization. In the original definition of PSO, the quantity of solutions analyzed remains fixed during the adaptive process [6] [11].

2 Objective

This paper aims at presenting a new extension of PSO, called VarPSO, incorporating the concepts of age and neighborhood to allow varying the size of the population. In this way, not only the quality of the solution to be obtained will be independent of the initial swarms size but also the commitment relationship existing between the convergence speed and the population diversity will be improved. The variation of the population size is based on a modification of the adaptive process allowing adding and/or deleting individuals in function of its capacity to solve the posed problem. This is carried out through the concept of age, which allows determining the time of permanence of each element within the population. In addition, since PSO tends to drive particles towards the explored areas with good fitness and in order to not excessively overpopulate certain solution spaces, each individuals environment is analyzed and the worst solutions of the areas with larger number of particles are eliminated.

The method here proposed, VarPSO, is applied to the resolution of some complex functions, finding better results than those typically achieved using a fixed size population.

This paper is organized as follows: Section 3 briefly describes the basic PSO algorithm; Section 4 presents the concepts necessary to carry out the variation of the swarms size; Section 5 presents in detail the proposed algorithm; Section 6 introduces the results obtained, and Section 7, the conclusions and future lines of work.

3 Swarm Particle-Based Algorithms

A Swarm Particle-based algorithm, also called Particle Swarm Optimization (PSO), is a heuristic population technique in which each individual represents a potential solution to the problem and makes its adaptation taking into account three factors: its knowledge on the environment (its fitness value), its historical background or previous experiences (its memory), and its historical background or previous experiences of its neighborhoods individuals [6]. Its objective consists in evolving its behavior so as to look like those most successful individuals within its environment. In this type of technique, each individual remains constantly moving within the search space and never dies. On its part, the population can be considered as a multi-agent system in which each individual or particle moves around the search space saving and, eventually, communicating the best solution found.

There exist various versions of PSO; the most known are gBest PSO, which uses as neighborhood criterion the overall population, and lBest PSO which, on the

other hand, makes use of a small sized population [6] [11]. The size of the neighborhood impacts on the algorithm convergence neighborhood as well as on the diversity of the populations individuals. When the neighborhood size increases, the convergence of the algorithm is faster but the diversity of individuals is lower. Each particle p_i is made up by three vectors and two fitness values:

- Vector $x_i = (x_{i1}, x_{i2}, , x_{in})$ stores the current position of the particle in the search space.
- Vector $pBest_i = (p_{i1}, p_{i2}, , p_{in})$ stores the best position of the solution found by the particle up to the moment.
- Speed vector $v_i = (v_{i1}, v_{i2}, , v_{in})$ stores the gradient (direction) according to which the particle will move.
- The fitness $valuefitness\_x_i$ stores the current solution capacity value (vector x_i).The fitness value $fitness_pBest_i$ stores the capacity value of the best local solution found up to the moment(vector $pBest_i$).

The position of a particle is updated as follows

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{1}$$

As previously explained, the speed vector is modified taking into account its experience and the environments. The expression is the following:

$$v_{ij}(t+1) = w.v_i(t) + \varphi_1.rand_1.(pBest_i - x_i(t)) + \varphi_2.rand_2.(g_i - x_i(t)) \tag{2}$$

where w represents the inertia factor [10], φ_1 and φ_2 are acceleration constants, $rand_1$ and $rand_2$ are random values belonging to the interval (0,1), and g_i represents the position of the particle with the best fitness of the environment of p_i (lBest o localbest) or the whole swarm (gBest o globalbest). Values of w, φ_1 and φ_2 are essential to assure the algorithms convergence. For more details on the selection of these values, consult [3] and [1].

4 Variable Population Size Particle Swarms

The variation of the populations size is accomplished by allowing the particle to reproduce and die during the adaptation process. This requires the definition of mechanisms regulating the corresponding insertion and elimination processes.

4.1 Life Time

One of the most important concepts of the proposed strategy is the particle's life time, since it determines the duration of its permanence within the population. Such value is expressed in quantity of iterations, which once over, the particle is removed. This value is closely related to the capacity of each particle and allows the best to remain longer in the population, influencing the behavior of the others.

In order to assess the life time of each population individual, the method of assignment by classes defined in [7] was used, since it has proved to be capable of

providing good results with a smaller quantity of individuals than that applied by conventional methods. In [7], individuals of a population are grouped according to their capacity value in k classes using a winner-take-all-type competitive clustering method. Over the result of this clustering, we can apply one of the following methods:

a) Fixed life time assignment by class. The maximum time of the life time to assign is divided by the quantity of classes, k. This allows us to know the time range corresponding to each class. Within a same class, its individuals will receive a life time proportional to the class to which they belong and to the quantity of individuals in the same class, as follows: $WidthClass := MAX\_LT/k$
$TVPrev := (NumC - 1) * WidthClass$
$TVCurrent := WidthClass$
$Displacement = (fitness[i] - Class[NumC].MinFit)/$
$abs(Class[NumC].MaxFit - Class[NumC].MinFit))$
$LifeTime[i] := trunc(TVPrev + TVCurrent * Displacement)$

where

- $WidthClass$ is the life time range assigned to each class.
- $NumC$ is the number of class to which each individual belongs.
- $TVCurrent$ is the life time range of the class to which the individual belongs.
- $TVPrev$ is the life time range assigned to the classes previous to $NumC$.
- $Class[NumC].MinFit$ and $Class[NumC].MaxFit$ are the values of minimum and maximum capacity of the class to which the individual under consideration belongs.
- $Fitness[i]$ is the value of the population i-th individuals capacity.

b) Life time assignment proportional to the quantity of each classs individuals. Each class receives a life time range proportional to the quantity of elements it contains. That is, individuals belonging to numerous classes could have a wider life time range. The computation is as follows:
$TotalPrev := 0$
for i :=1 to NumC - 1 do TotalPrev := TotalPrev + Class[i].Cant
$TVPrev := MAX\_LT * TotalPrev/TotalIndiv$
$TVCurrent := MAX\_LT * Class[NumC].Cant/TotalIndiv$
$Displacement = (fitness[i] - Class[NumC].MinFit)/$
$abs(Class[NumC].MaxFit - Class[NumC].MinFit))$
$LifeTime[i] := trunc(TVPrev + TVCurrent * Displacement)$

where

- $Class[NumC].Cant$ represents the total quantity of individual of the closest class.
- $TotalIndiv$ is the total quantity of the individuals of the population.

These two ways of computing the individuals life time should be combined in order to achieve the proper assignment. We propose to apply assignment b)

during acertain percentage of the algorithms maximum generation quantity and apply a) in the remaining. This is due to the fact that initial clustering is carried out over individuals which are not completely adapted yet and, thus, they give place to highly dissimilar sized clusterings. If we directly apply the distribution indicated in a) over these clusterings, several individuals will received similar life times, leading the algorithm to unnecessarily increase the quantity of individuals in the population.

4.2 Particle Insertion

Particle insertion has two objectives: increasing the convergence speed by incorporating individuals in the less populated areas and compensating the particle elimination caused by the fulfillment of the corresponding life times. Determining the convenient locations, within the search space, where the new individuals should be inserted is not a trivial task. In fact, it is a commitment between the optimal area identification and the new individuals insertion process speed.

The adopted solution divides the original problem in two parts: first, it seeks to determine how many particles it is necessary to incorporate so as to then establish where they should be placed within the search space.

The quantity of incorporated particles at each iteration coincides with the quantity of *isolated* individuals. An *isolated* individual is that which does not have any neighbor within a pre-established r radius. Equations 3 and 4 show the way in which such radius should be computed [2]. As it can be seen, r is computed as the average of each particles distances with their closest neighbor.

$$d_i = min\{\|x_i - x_j\|; \forall j x_i, x_j \epsilon S; x_i \neq x_j\} \qquad i = 1..n \tag{3}$$

$$r = \frac{\sum_{i=1}^{n} d_i}{n} \tag{4}$$

It only remains to determine the position of these new individuals. The adopted criterion was the following: the 20 % of these new particles receive position vector of the best individuals of the population, but its speed vector is random; the remaining 80% is random. In this way, part of the new individuals will begin to move from the positions that have shown better performance up to the moment, but with different directions and speeds from those of the best individuals. The remaining 80% will allow exploring other areas of the search space.

It is important to notice that the efficacy of the distance measure used in 3 will depend on the selected search space representation. If necessary, you may consult other alternatives in [4].

5 Proposed Algorithm

The algorithm begins with apopulation of N individuals generated at random within the search space and computes, for each of them, their corresponding fitness and life time.

During the process, individuals move according equations 1 and 2.The inertia used in order to update speed vectors is adjusted as follows [8]:

$$w = w_{start} - \frac{(w_{start} - w_{end})}{TotalIteraciones}.CurrentIteration \quad (5)$$

where w_{start} is the initial value of w and w_{end} is the end value.

A high value of w at the beginning of the evolution allows the particles to make large movements placing themselves in different position of the search space. As the number of iterations advances, the value of w is reduced, allowing them to make a finer adjustment.

From the new positions within the search space, the individuals fitness value is recomputed and radius r is obtained according to equation 4.

Then, as many individuals as particles exist in the population without neighbors within this radius are created. These new individuals will have random speed vectors, within the allowed ranges. The 20% of these new particles will receive the position vectors of the best individuals of the population, and the remaining 80% will have random position vectors. For these new particles, their fitness is assessed and they are then incorporated to the population. Using the complete population, the recently incorporated individuals life time is computed.

The life time of every individual is decreased in 1, and those who have reached zero value are eliminated from the population.

The proposed algorithm makes use of elitism, reason why the best individual of each iteration is preserved. In this way, it is assured that the population will have at least one particle. This is carried out by replacing the particle with smaller fitness by the best one of the previous iteration.

Finally, the algorithm ends when one of the following conditions is met:

- The initially indicated maximum quantity of iteration is reached.
- The best fitness has not been modified during the 15% of the total iterations.

Figure 1 presents the pseudo-code of the described algorithm.

The *CreatePopulation* function receives as parameter the quantity of particles to be created and returns a swarm with random position and speed vectors within the established limits and with null life times. In order to estimate them, it is necessary to first assess each individuals fitness.

The *ComputeLifeTimes* process receives a complete swarm and only computes the life time corresponding to the particles that have null life time at the moment of the invocation. The second parameter corresponds to the type of computation that should be carried out and it values 1 for the assignment described in 4.1.a), and 2 for that described in 4.1.b).

The radius computation,according to equation 4, is carried out within the *ComputeRadius* process which receives as parameter the complete swarm and returns the quantity of new particles that should be inserted in the population. This module is the one in charge of avoiding the concentration of several particles in the same place of the search space; for this reason, it also returns the list of individuals that have really closed neighbors. Such particles are eliminated in

the *SeeEnvironmnet* module, in function of their fitness and the quantity of generated sons.

Pop= CreatePopulation(N)
ComputeFitness(Pop);
ComputeLifeTimes(Pop,2);
w←MAXIMUMINTERTIA
while no end condition is reached do
  for i = 1 to size(S) do
    evaluate particle x_i of swarm S
    if fitness(x_i) is better than $fitness(pBest_i)$ then
      $pBest_i \leftarrow x_i$; $fitness(pBest_i) \leftarrow fitness(x_i)$
    end if
  end for
  Save individual with maximum fitness.
  for i = 1 to size(S) do
    Choose g_i according to neighborhood criterion used
    $v_i \leftarrow w.v_i + (\varphi_1.rand_1.(pBest_i - x_i) + \varphi_2.rand_2.(g_i - x_i)$
    $x_i \leftarrow x_i + v_i$
  end
  ComputeFitness(Pop);
  ComputeRadius(Pop, SonQuant, Sentenced);
  New = CreatePopulation(SonQuant);
  Assign 20% to these new individuals the position vectors of the best individuals of Pop.
  ComputeFitness(New);
  SeeEnvironment(Pop, Sentenced, SonQuant);
  Pop = Pop ∪New;
  if (CurrentIteration is greater than 5%of the TOTALITERATIONS))
      ComputeLifeTimes(Pop, 1);
    else ComputeLifeTimes(Pop, 2);
  end
  Deduct 1 to each particles life time
  Remove the particles with null life time
  Replace the worst individual by the saved at the beginnning of this iteration
  w ← dynamically modify the inertia
end while
Output : the best solution found

Fig. 1. Algorithm of the proposed VarPSO method

6 Results Obtained

VarPSO was used in order to obtain the minimum value of several functions. Thus, each particle contains in its position vector the values of the function statements. The capacity of each particle is computed as follows:

$$(c\_max - Value\_of\_the\_Particle) \tag{6}$$

where *c_max* represents the function top limit in the interval to be optimized, and *Value_of_the_Particle* is the result from assessing the function in the corresponding particles position vector.

Next, the functions used are presented in detail. For each of them, the interval used to determine the search space is shown together with the *c_max* value used to compute the fitness.

$$F1(x,y) = x^2 + y^2 \qquad x,y \in [-1,5]; c\_max = 50$$

$$F2(x) = -x * sin(10 * \pi * x) + 1 \qquad x \in [-2,1]; c\_max = 3$$

$$F3(x,y) = 0.5 + \frac{(sin(\sqrt{x^2+y^2+4}))^2 - 0.5}{(1+0.001.(x^2+y^2))^2} \qquad x,y \in [-50,50]; c\_max = 1$$

$$F4(x_1,x_2) = \frac{1}{0.002 + \sum_{j=1}^{25} \frac{1}{50j + \sum_{i=1}^{2}(x_i - a_{ij})^6}} \qquad x_1,x_2 \in [-50,50]; c\_max = 500$$

$$F5(x_1,x_2) = \frac{1}{0.002 + \frac{1}{1+(x_1+1)^{12}+(y_1+1)^{12}}}) \qquad x_1,x_2 \in [-200,200]; c\_max = 500$$

$$F6(x_1,x_2) = \frac{1}{0.002 + \sum_{j=1}^{3} \frac{1}{50j^2 + \sum_{i=1}^{2}(x_i - b_{ij})^{12}}} \qquad x_1,x_2 \in [-50,50]; c\_max = 500$$

$$b_{ij} = \begin{pmatrix} -30 & 16 & 30 \\ -40 & -32 & 35 \end{pmatrix}$$

The six functions present a single minimum. Each of them seeks measuring a different aspect of the proposed algorithm. F1 allows analyzing the precision of the solution obtained. F2, F3, and F4 show their capacity of moving along a search space with really changing fitness values. Notice that function F4 is a modification of the De Jong's function 5. Functions F5 and F6 have been introduced to analyze the algorithm's exploratory capacity. In F5 a single gap appears, which leads to the minimum within a completely flat surface. F6 is similar, but it presents three gaps with very different depths.

For each function, 100 tests were carried out, using in each case, a maximum quantity of 500 iterations. The used cognitive and social learning values, φ_1 and φ_2, described in equation 2, were both in 0.5. Inertia values were between 0.2 and 1.5. The allowed speed range was established between -0.5 and 0.5.

The quantity of classes used for computing the individuals life time was 4. Tests were carried out with values between 2 and 10, confirming that a high value for the number of classes improves the fitness distinction among individuals, but significantly increases the population size. On the other hand, if the class quantity is very low, the swarms size could be considerably decreased. A value of 4 classes is adequate for the minimization of the afore mentioned functions. The value used as maximum life times was of 9 iterations, except in functions F5 and F6, which used a value of 12.

Table 1. Results obtained with PSO and VarPSO

Function F1	Avg.Ite.	Min. Fit.	Med. Fit.	Max. Fit.	Ini.Pop	Fin.Pop
gBest PSO	106.90	48.0544	49.2226	50.0000	20	20.0
lBest PSO	142.08	48.0373	49.2735	50.0000	20	20.0
gBestVarPSO	181.89	30.4806	45.3319	49.9996	30	33.8
lBestVarPSO	169.82	30.3254	45.0502	49.9995	30	32.8
Function F2	Avg.Ite.	Min. Fit.	Med. Fit.	Max. Fit.	Ini.Pop	Fin.Pop
gBest PSO	102.67	1.6791	2.1150	3.8466	60	60.0
lBest PSO	102.40	1.7325	2.0647	3.8456	70	70.0
gBestVarPSO	115.22	1.3443	2.3861	3.8489	40	13.1
lBestVarPSO	122.18	1.2349	2.3705	3.8489	30	17.7
Function F3	Avg.Ite.	Min. Fit.	Med. Fit.	Max. Fit.	Ini.Pop	Fin.Pop
gBest PSO	231.06	0.7288	0.9074	0.9762	70	70.0
lBest PSO	197.59	0.6130	0.8607	0.9717	60	60.0
gBestVarPSO	175.17	0.3032	0.5857	0.9942	20	53.3
lBestVarPSO	154.88	0.3025	0.5866	0.9936	40	62.6
Function F4	Avg.Ite.	Min. Fit.	Med. Fit.	Max. Fit.	Ini.Pop	Fin.Pop
gBest PSO	284.10	441.7729	445.3109	446.6185	60	60.0
lBest PSO	218.77	437.8757	443.5489	444.9094	70	70.0
gBestVarPSO	132.04	145.0876	285.1007	453.0275	30	30.7
lBestVarPSO	124.64	156.9515	299.9142	454.5333	40	55.0
Function F5	Avg.Ite.	Min. Fit.	Med. Fit.	Max. Fit.	Ini.Pop	Fin.Pop
gBest PSO	140.19	468.3641	482.4536	484.0312	70	70.0
lBest PSO	138.37	448.4032	477.8286	479.0419	70	70.0
gBestVarPSO	117.01	171.4216	280.7481	494.0287	40	16.2
lBestVarPSO	121.72	134.1485	238.5021	496.0285	30	23.8
Function F6	Avg.Ite.	Min. Fit.	Med. Fit.	Max. Fit.	Ini.Pop	Fin.Pop
gBest PSO	103.27	373.4571	391.0167	391.8489	60	60.0
lBest PSO	115.49	404.7819	440.9218	442.8571	50	50.0
gBestVarPSO	92.56	71.9364	194.3381	443.2076	40	85.7
lBestVarPSO	108.44	120.5613	228.8209	443.4110	20	52.9

Table 1 allows us to compare the results obtained when applying the algorithm proposed in this paper and the fixed population PSO algorithm. Tests were carried out with initial population size of 5, 10, 20, 30, 40, 50, 60, and 70 particles. In all the cases, the values correspond to the averages of the 100 tests carried out for each function, taking as initial population size that which allowed obtaining the best maximum average fitness.

As it can be seen, in all the cases, the solution obtained using VarPSO is superior or equal to that reached by PSO with fixed population, presenting greater population diversity. In addition, except function F1 for which, it is quite simple to reach the maximum fitness, the proposed method makes use of a inferior initial population than that of the fixed population method. If we analyze the quantity of particles displaced in average in each case, we can see that, in most of the functions, the proposed method carries out less than half of the job

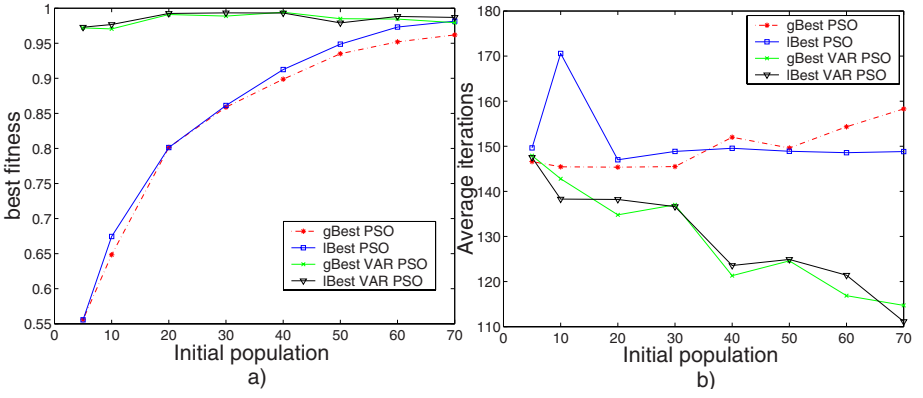

Fig. 2. a) Average maximum fitness obtained for different initial population sizes. b) Variation of the average iteration number necessary to obtain the best fitness in function of the initial population size.

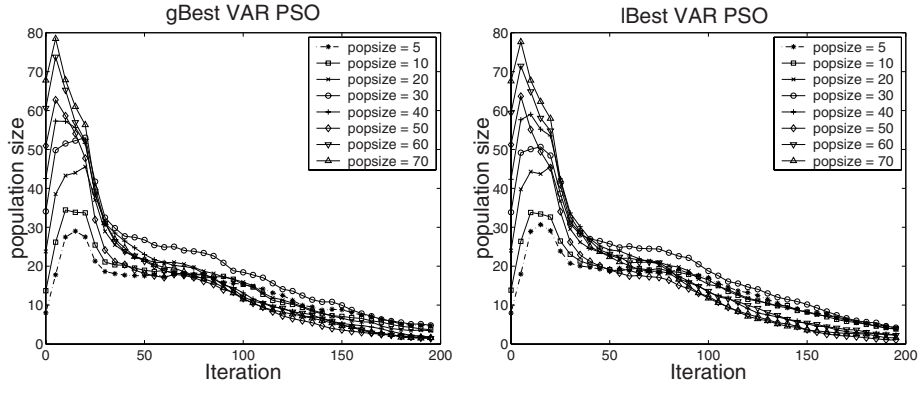

Fig. 3. Average size of the population using gBest VarPSO and lBest VarPSO. The values correspond to the first 200 iterations average of the 600 tests carried out (100 over each function).

than that carried out by fixed population solutions. This last is not verified for functions F1 and F6. In the case of F1, it is due to the functions simplicity, which contrasts with the exploratory behavior of the proposed method, and in the case of F6, it is due to the fact that the method does not prematurely converge like its fixed population peer, but goes on analyzing the search space obtaining best solutions.

Figure 2.a) shows the capacity of the proposed method to adapt itself to the searching surface, finding an almost optimal solution independent of the size of the initial population. We can also observe that fixed population methods provide proper results when the adequate initial population size is used. Each point of figure 2.a) corresponds to the maximum average fitness of the 600 tests

(100 for each function) carried out for each initial population size. In each case, the fitness has been linearly scaled to [0,1], dividing it by the maximum fitness corresponding to the function.

If we analyze the average quantity of iterations carried out for each method in function of the size of the initial population, we can see that, for fixed population solutions, it tends to slightly increase while for variable size populations, it rapidly decreases as the initial population increases. This is represented in Figure 2.b) and reflects the behavior of each method. Fixed population methods only depends on the initial particle displacement, while variable population methods carry out an initial increase of the population which allows them to rapidly explore a wider area of the search space, reaching at the optimum with a smaller number of iterations. Finally, Figure 3 shows the average growth of the population for both variants of the proposed method. As it can be seen, the behavior is rather similar in both cases and presents a growth phase, within the first 30 iterations, followed by a reduction and stabilization phase.

7 Conclusions and Future Works

A strategy based on particle swarm (PSO) with variable size population based on the concepts of age and neighborhood has been presented. It has been proven that the mechanism used to incorporate new individuals and the way in which life time is computed preserves the diversity of the population. Its application on different complex functions has been compared to gBestPSO and lBest PSO. The results obtained show that the strategy proposed in this paper allows obtaining results with a better capacity value than those obtained with both version of PSO with fixed population size, using a smaller quantity of iterations.

Work is currently under way on the parallelization of the proposed algorithm using only the information environment of each particle to determine whether this is an isolated individual or not. Also algorithm migration to cluster/grid arquitectures is analyzed. Moreover, according to the great exploration capacity shown by method proposed in this paper, we are currently working on its applications to neural networks' adaptation. In this case, each particle represents a complete feedforward neural network with fixed architecture and, along iterations, we intend to adapt the connection weights. It remains defining a similarity measure assuring the proper quantification of the proximity between two individuals in the search space.

Acknowledgment

This research was partially funded by the project CyTED "Grid Technology for Boosting Regional Development".

References

1. Van den Bergh, F.: An analysis of particle swarm optimizers. Ph.D. dissertation. Department Computer Science. University Pretoria. South Africa (2002)
2. Bird, S., Li, X.: Adaptively Choosing Niching Parameters in a PSO. In: Keijzer, M., et al. (eds.) Proceeding of Genetic and Evolutionary Computation Conference 2006 (GECCO 2006), pp. 3–9. ACM Press, New York (2006)
3. Clerc, M., Kennedy, J.: The particle swarm-explosion, stability and convergence in a multidimensional complex space. IEEE Transactions on Evolutionary Computation 6(1), 58–73 (2002)
4. Ertöz, L., Steinbach, M., Kumar, V.: A new shared nearest neighbor clustering algorithm and its applications. In: Proc. Workshop on Clustering High Dimensional Data and its Applications, Arlington, VA, USA, pp. 105–115 (2002)
5. Fernandes, C., Ramos, V., Rosa, A.: Varying the Population Size of Artificial Foraging Swarms on Time Varying Landscapes. In: Duch, W., Kacprzyk, J., Oja, E., Zadrożny, S. (eds.) ICANN 2005. LNCS, vol. 3696, pp. 311–316. Springer, Heidelberg (2005)
6. Kenedy, J., Eberhart, R.: Particle Swarm Optimization. In: Proceedings of IEEE International Conference on Neural Networks, Australia, vol. IV, pp. 1942–1948 (1995)
7. Lanzarini, L., Sanz, C., Naiouf, M., Romero, F.: Mixed alternative in the assignment by classes vs. conventional methods for calculation of individuals lifetime in GAVaPS. In: Proceedings of the 22nd International Conference on Information Technology Interfaces, 2000. ITI 2000, pp. 383–389 (2000); ISBN: 953-96769-1-6.
8. Meissner, M., Schmuker, M., Schneider, G.: Optimized Particle Swarm Optimization (OPSO) and its application to artificial neural networks training. In: BMC Bioinformatics 2006 (Published online 2006 March 10) pp. 7–125 (2006) DOI: 10.1186/1471-2105-7-125
9. José, G.N.: Algorithms based on swarms of particles for solving complex problems. University Málaga (In Spanish) (2006)
10. Shi, Y., Eberhart, R.: Parameter Selection in Particle Swarm Optimization. In: Proceedings of the 7th International Conference on Evolutionary Programming, pp. 591–600. Springer, Heidelberg (1998)
11. Shi, Y., Eberhart, R.: An empirical study of particle swarm optimization. In: Proceeding on IEEE Congress Evolutionary Computation, Washington DC, pp. 1945–1949 (1999)