Journal of Computer Science 4 (2): 139-147, 2008
ISSN 1549-3636
© 2008 Science Publications

Memory Convergence and Optimization with Fuzzy PSO and ACS

[1]Subhash Chandra Pandey and [2]Dr. Puneet Misra
[1]Birla Institute of Technology, Extension Centre: Naini, Allahabad, 211010, India
[2] Department of Computer Science, Lucknow University, Lucknow, India

Abstract: Associative neural memories are models of biological phenomena that allow for the storage of pattern associations and the retrieval of the desired output pattern upon presentation of a possibly noisy or incomplete version of an input pattern. In this study, we introduce fuzzy swarm particle optimization technique for convergence of associative neural memories based on fuzzy set theory. A Fuzzy Particle Swarm Optimization (FPSO) consists of clustering of swarm's particle by applying fuzzy c-mean algorithm to attain the neighborhood best. We present a singular value decomposition method for the selection of efficient rule from a given rule base required to attain the global best. Finally, we illustrate the proposed method by virtue of some examples. Further, ant colony system ACS algorithm is used to study the Symmetric Traveling Salesman Problem TSP. The optimum parameters for this algorithm have to found by trial and error. The ACS parameters working in a designed subset of TSP instances has also been optimized by virtue of Particle Swarm Optimization PSO.

Key words: Artificial neural network, convergence, particle swarm optimization, Fuzzy c-mean

INTRODUCTION

An artificial neural network (ANN) is an analysis paradigm that is a simple model of the brain and the back-propagation algorithm is the one of the most popular method to train the artificial neural network. Recently there have been significant research efforts to apply evolutionary computation techniques for the purposes of evolving one or more aspects of artificial neural networks.

The efficient supervised training of feedforward neural networks (FNNs) is a subject of considerable ongoing research and numerous algorithms proposed to this end. The back propagation (BP) algorithm[1] is one of the most common supervised training methods. Although BP training has proved to be efficient in many applications, its convergence tends to be slow and yields to suboptimal solutions[2].

Attempts to speed up training and reduce convergence to local minima have been made in the context of gradient descent[3,4,5]. However, these methods are based on variable weight, learning rate, step size and bias to dynamically adapt BP algorithm and use a constant gain for any sigmoid function during its training cycle.

Evolutionary computation methodologies have been applied to three main attributes of neural networks: network connection weights, network architecture (network topology, transfer function) and network learning algorithm.

Particle swarm optimization (PSO) is a population based stochastic optimization technique[6,7] inspired by social behavior of bird flocking or fish schooling. This is modeled by particles in multidimensional space that have a position and a velocity. These particles are flying through a hyperspace and have two essential reasoning capabilities: the memory of their own best position and knowledge of the swarm's best, best simply meaning the position with the smallest objective function value. Members of a swarm communicate good positions to each other and adjust their own position and velocity based on good positions. There are two main ways this communication is done:

- A global best that is known to all and immediately updated when a new best position is found by any particle in the swarm.
- A Neighborhood best where each particle only immediately communicates with a subset of the swarm about best positions.

In this study we have designed an algorithm using a Particle Swarm Optimization (PSO) framework, to optimize the parameters of the ACS algorithm working

Corresponding Author: Subhash Chandra Pandey, Department of Computer Science, Birla Institute of Technology, Extension Centre: Naini, Allahabad, India-211010

on a single Symmetric Travelling Salesman Problem (TSP) instance. For each instance the algorithm computes an optimal set of ACS parameters, their performance on all instances (not only their related instance) and the characteristics of their related instance for the purpose of finding correlations.

The first ACO algorithm, called Ant-System, was proposed in[8-10]. A full review of ACO algorithms and applications can be found in[11]. ACS is a version of the Ant System that modifies the updating of the pheromone trail[12,13]. We have chosen this ACS algorithm to work with because of the theoretical background we have found on it[11,12] and the previous fine-tuning research on the parameters by[14].

We have chosen PSO because it has an easy implementation for integer and real parameters and, as genetic algorithms, it performs a blind search on all the possible sets of parameters. In our algorithm the domain of the PSO will be all possible sets of parameters for ACS. For a position of a particle we compute the fitness by running the ACS algorithm with the parameters given by the position on a TSP instance.

THE PARTICLE SWARM OPTIMIZATION

PSO's precursor was a simulator of the social behavior that was used to visualize the movement of a birds' flock. Several version of the simulation model were developed, incorporating concepts such as nearest neighbor velocity matching and acceleration by distance[6,15]. Two variants of the PSO algorithm were developed, One with a global neighborhood and another with local neighborhood[16].

Suppose that the search space is *D*-dimensional and then the ith particle of the swarm can be represented by a D-dimensional vector $X_i = (x_{i1}, x_{i2},...,x_{iD})$. The velocity (position change) of this particle can be represented by another D-dimensional vector $V_i = (v_{i1}, v_{i2},...,v_{iD})$. The best previously visited position of the ith particle is denoted as $P_i = (p_{i1}, p_{i2},...,p_{iD})$. Defining *g* as the index of the best particle in the swarm (i.e., gth particle is the best) and let the superscript denote the iterative number, then the swarm is manipulated according to the following two equations[6]:

$$z_{id}^{n+1} = z_{id}^{n} + Cr_1^n(p_{id}^n - x_{id}^n) + Cr_2^n(p_{id}^n - x_{id}^n) \qquad (1)$$

$$x_{id}^{n+1} = x_{id}^{n} + z_{id}^{n+1} \qquad (2)$$

where, d = 1,2,...D, i = 1,2,...,N and N is the size of the swarm, C is a positive constant called, acceleration constant r_1, r_2 are the random numbers, uniformly distributed in [0,1], and n = 1,2,…,determines the iteration numbers.

Equations 1 and 2 define the initial version of the PSO algorithm. Since there was no actual mechanism for controlling the velocity of a particle, it was necessary to impose a maximum value V_{max} on it. If the velocity exceeded this threshold, it was set equal to V_{max}. This parameter is proved to be crucial, because large values could results in the particles moving past good solutions, while small values could result in insufficient exploration of the search space. This lack of control mechanism for the velocity resulted in low efficiency for PSO.

Various attempts have been made to improve the performance of the base line PSO with varying success. In[16] emphasis has been given on optimizing the update equations for the particles. Some researcher used a selection mechanism in an attempt to improve the general quality of the particles in swarm. In[6] cluster analysis technique is used to modify the update equation, so that particles attempt to confirm to the centre of their clusters rather than attempting to conform to a global best.

The aforementioned problem was addressed by incorporating a weight parameter for the previous velocity of the particle. Thus in the latest version of the PSO, Eq. 1 and 2 are changed to the following ones[17]:

$$x_{id}^{n+1} = X_{id}^{n} + x_{id}^{n+1}\left(wZ_{id}^{n} + C_1 r_1^n(p_{id}^n - X_{id}^n) + C_2 r_2^n(p_{id}^n - X_{id}^n)\right)$$

$$x_{id}^{n+1} = X_{id}^{n} + x_{id}^{n+1}$$

In our proposed model both the approaches have been consider together. First, we clustered the swarm by applying fuzzy c-mean algorithm to attain the neighborhood best and then we reduce the number of rules required to attain the global best by virtue of singular value decomposition method.

NEIGHBORHOOD BEST USING FCM

The Fuzzy C-Means algorithm generalizes the hard C-means algorithm to allow a particle of swarm to partially belong to a multiple clusters. Therefore, it produces a soft partition for a given swarm. To do this, the objective function J of hard c-means has been extended in two ways Fig 1 and 2.

The fuzzy membership degrees in clusters were incorporated into the formula and:

- An additional parameter P is introduced as a weight exponent in the fuzzy membership.

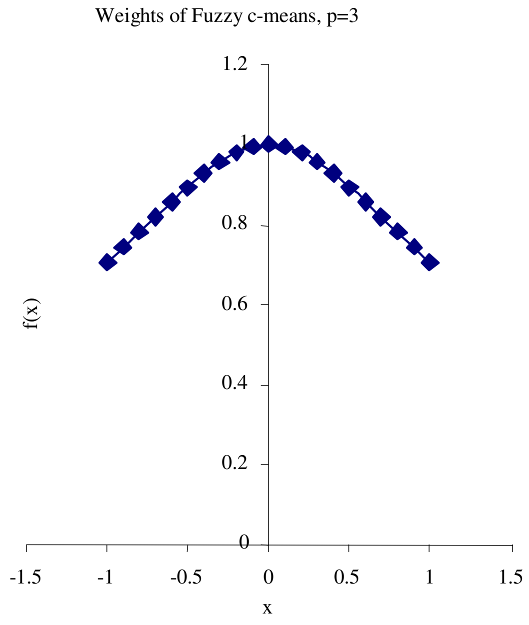

Fig.1: Weight of the FCM algorithm for p = 3

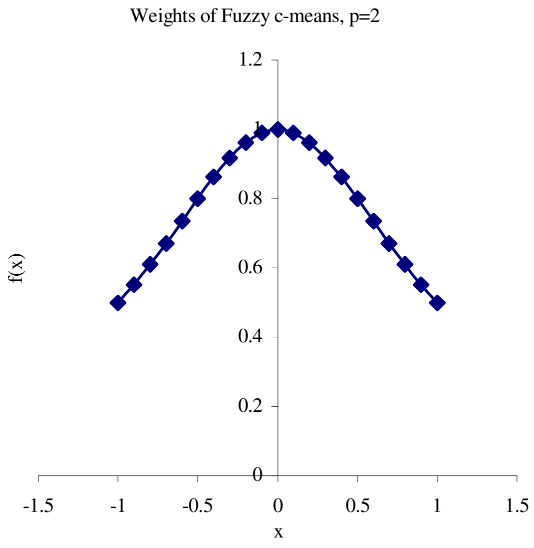

Fig.2. Weight of the FCM algorithm for p = 2

- The extended objective function, denoted J, is

$$J(P,V)=\sum_{i=1}^{k}\sum_{X_k\in X}(\mu_c(X_k))^p\|X_k-V_i\|^2$$

where, P is a fuzzy partition of the swarm X formed by $C_1, C_2,...,C_k$. The parameter p is a weight that determines the degree to which partial members of a cluster effect the clustering result.

Theorem: A constrained fuzzy partition $(C_1, C_2,...,C_k)$ can be a local minimum of the objective function J only if the following conditions are satisfied:

$$\mu_{ci}(X)=1\Big/\sum_{j=1}^{k}\left(\|X-V_i\|^2/\|X-V_j\|^2\right)^{1/p-1} \qquad (3)$$

$$V_i=\sum_{x\in X}(\mu_{ci}(x))^p\times x\Big/\sum_{x\in X}(\mu_{ci}(x))^p \qquad (4)$$

Based on this theorem, FCM updates the prototypes and the membership function iteratively using 3 and 4, until a convergence criterion is reached.

The algorithm of FCM can be described as:

FCM (X, c, m, ε)

X: An unlabeled swarm size
C: The number of clusters to form
p: The parameter in objective function
ε: A threshold for the convergence criteria.

Initial prototype $V = \{v_1, v_2,...,v_c\}$

Repeat
$V^{previous} \leftarrow V$ Compute membership function using 4
Update the prototype, v_i in using 3 Until $\sum_{i=1}^{c}\left\|V_i^{previous}-V_i\right\|\le\varepsilon$

GLOBAL BEST USING SVD

In our proposed model, after clustering the particles of swarm, orthogonal transformation method is used for selecting important fuzzy rules from a given rule base[18-21]. Unlike conventional methods where multiple iterations are usually required to find optimal number of fuzzy rules, orthogonal transformation methods are a non iterative procedure. Therefore, orthogonal transformation methods are computationally less expensive compared to the conventional methods especially when the numbers of particles in the swarm are too large. In this section we introduce how to use singular value decomposition (SVD) to select the most important fuzzy rules from a given rule base and construct compact fuzzy models with better generalization ability.

Singular value decomposition takes a rectangular *n*-by-*p* matrix *A*, in which the *n* rows represents the genes and the columns represents the experimental condition[22]. The SVD theorem states:

$$A_{n\times p}=U_{n\times n}S_{n\times p}V^T_{p\times p}$$

Where $U^TU = I_{n\times n}$

$V^TV = I_{p\times p}$ (i.e. U and V are orthogonal)

S = diag ($\sigma_1, \sigma_2, \sigma_3,..,\sigma_m$)$\in R^{n\times p}$ (m = min{n,p}) is a diagonal matrix with $\sigma_1\geq\sigma_2\geq\sigma_3\geq\ldots\geq\sigma_m\geq 0$. The columns of U are the left singular vectors has singular values and is diagonal (mode amplitudes), and V^T has rows that are the right singular vectors (expression level vectors).

In the basic principle of using SVD for fuzzy rule selection, we can use fuzzy model with constant consequent constituents as an example. This type of fuzzy model, which is usually referred to as the zero order TSK model, has the following form[23].

R_i: If x_1 is A_{i1} and x_2 is A_{i2} and ………andxm is A_{im} Then y is C_i, i = 1,2,…..,M.

Where, C_i is the constant constituents. The total output of the model is computed by.

$$Y = \sum_{i-1}^{M} w_i c_i / \sum_{i-1}^{M} w_i$$

where, w_i is the matching degree.

The SVD starts with an oversized rule base and then remove redundant or less important fuzzy rules through a one pass operation. Finally the efficient rule obtained is obeyed by all the swarm's cluster to approach the global best. Now we will illustrate the method by taking a example.

Suppose we are given a swarm of size six particles, each of which has two features F_1 and F_2.We list the particle in given Table 1. Assuming that we want to use FCM to partition the swarm in two clusters[23], suppose we set the parameter p in FCM at 2 and the initial prototypes to $v_1 = (5,5)$ $v_2 = (10,10)$.

The initial membership functions of the two clusters are calculated using 3:

$$\mu_{ci}(x) = 1 / \Sigma^2_{j=1}(\| x_1-v_i \| / \| x_1 - v_j \|)^2$$
$$\| x_1-v_1 \|^2 = 3^2 + 7^2 = 9 + 49 = 58$$
$$\| x_1-v_2 \|^2 = 8^2 + 2^2 = 64 + 4 = 68$$
$$\mu_{c1}(x_1) = 1/ [(58/58) + (58/68)]$$

Table 1: A swarm to be partitioned

	F_1	F_2
X1	2	12
X2	4	9
X3	7	13
X4	11	5
X5	12	7
X6	14	4

Similarly, we obtain the following:

$$\mu_{c2}(x_1) = 1/[(68/58)+(68/68)] = 0.4603$$
$$\mu_{c1}(x_2) = 1/[(17/17)+(17/37)] = 0.6852$$
$$\mu_{c2}(x_2) = 1/[(37/17)+(37/37)] = 0.3148$$
$$\mu_{c1}(x_3) = 1/[(68/68)+(68/18)] = 0.2093$$
$$\mu_{c2}(x_3) = 1/[(18/68)+((18/18)] = 0.7907$$
$$\mu_{c1}(x_4) = 1/[(36/36)+(36/26)] = 0.4194$$
$$\mu_{c2}(x_4) = 1/[(26/36)+(26/26)] = 0.5806$$
$$\mu_{c1}(x_5) = 1/[(53/53)+(53/13)] = 0.197$$
$$\mu_{c2}(x_5) = 1/[(13/53)+(13/13)] = 0.803$$
$$\mu_{c1}(x_6) = 1/[(82/82)+(82/52)] = 0.3881$$
$$\mu_{c2}(x_6) = 1/[(52/82(+(52/52)] = 0.6119$$

Therefore, using three prototypes of the two clusters, the membership function indicates that x_1 and x_2 are more in the first cluster, while the remaining particles in the swarm are more in the second cluster.

The FCM algorithm then updates the prototypes according to 4.

$$V_1 = \Sigma^6_{k=1} (\mu_{ci}(x_k))^2 \times x_k / \Sigma^6_{k=1} (\mu_{ci}(x_k))^2$$

$$= [0.5397^2\times(2.12)+0.6852^2\times(4.9)+0.2093^2\times(7.13)+0.4194^2\times(11.5)+0.197^2\times(12.7)+0.3881^2\times(14.4)]/[0.5397^2+0.6852^2+0.2093^2+0.4194^2+0.197^2+0.3881^2]$$

$$= [(7.2761/1.0979), (10.044/1.0979)]$$
$$= (6.6273, 9.1484)$$

$$V_2 = \Sigma^6_{k=1}(\mu_{c2}(x_k))^2\times x_k / \Sigma^6_{k=1}(\mu_{c2}(x_k)0^2$$

$$[0.4603^2\times(2.12)+0.3148^2\times(4.9)+0.7909^2\times(7.13)+0.5806^2\times(11.5)+0.803^2\times(12.7)+0.6119^2\times(14.4)]/[0.4603^2+0.3148^2+0.7909^2+0.5806^2+0.803^2+0.6119^2]$$
$$= [(22.326/2.2928), (19.4629/2.2928)]$$
$$= (9.7374, 8.4887)$$

The update prototype V_1, as shown in Fig. 3, is

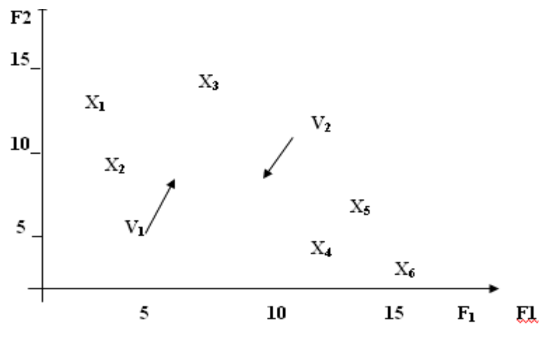

Fig.3 An example of Fuzzy c-mean Algorithm

moved closer to the center of the cluster formed by X_1, X_2 and X_3, while the updated prototype V_2 is moved closer to the cluster formed by X_4, X_5 and X_6.

Now we illustrate how to solve for SVD to obtain efficient rule for approaching the global best, let's take the example of the matrix.

$$A = \begin{bmatrix} 2 & 4 \\ 1 & 3 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

For a n×n matrix W, the nonzero vector X is the eigenvector of W if: $WX = \lambda X$, λ is the eigenvalue of A and X is the eigenvector of A corresponding to λ. So to find the eigenvalues of the entity we compute matrices AA^T and A^TA. The eigenvectors of AA^T make up the columns of U so we can do the following analysis to find U.

$$AA^T = \begin{bmatrix} 2 & 4 \\ 1 & 3 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 2 & 4 & 0 & 0 \\ 1 & 3 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 20 & 14 & 0 & 0 \\ 14 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Since $WX = \lambda X$, then $(W-\lambda I)X = 0$. Hence

$$\begin{bmatrix} 20-\lambda & 14 & 0 & 0 \\ 14 & 10-\lambda & 0 & 0 \\ 0 & 0 & -\lambda & 0 \\ 0 & 0 & 0 & -\lambda \end{bmatrix} X = (W-\lambda I)X = 0$$

Thus, from the solution of characteristic equation, we obtain $\lambda = 0$, $\lambda = 0$, $\lambda = 15+14.81$, $\lambda = 15-14.81$. This value can be used to determine the eigenvector that can be placed in the columns of U. Thus, we obtain the following equations.

$$19.883\,X1 + 14X2 = 0,\ 14X1 + 9.883\,X2 = 0,\ X3 = 0,\ X4 = 0$$

Upon simplifying the first two equations we obtain a ratio which relates the value of X1 and X2. The values of X1 and X2 are chosen such that the elements of S are the square roots of the eigenvalues. Thus a solution that satisfies the above equation X1 = -0.58 and X2 = 0.82 and X3 = X4 = 0 (this is the second column of the U matrix). Substituting the other eigenvalues we obtain:

-9.883X1+14X2 = 0, 14X1-19.883X2 = 0, X3 = 0, X4 = 0. Thus a solution that satisfies this set of equations is X1 = 0.82 and X2 = -0.58 and X3 = X4 = 0 (this is the first column of the U matrix). Combining these we obtain:

$$U = \begin{bmatrix} 0.82 & -0.58 & 0 & 0 \\ 0.58 & 0.82 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Similarly, we can find the value of V

$$A^TA = \begin{bmatrix} 2 & 4 & 0 & 0 \\ 1 & 3 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2 & 4 \\ 1 & 3 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Similarly, we obtain the expression

$$V = \begin{bmatrix} 0.40 & -0.91 \\ 0.91 & 0.40 \end{bmatrix}$$

Finally, the S is square root of the eigenvalues from AA^T or A^TA and can be obtained directly giving us:

$$S = \begin{bmatrix} 5.47 & 0 \\ 0 & 0.37 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

It is obvious that $\sigma_1 > \sigma_2 > \sigma_3 \ldots$ This is what the study was indicating.

ANT COLONY SYSTEM

The ACS works as follow, it has a population of n ants. Let denote for each arc e = (i, j) in the TSP-instance graph an initial heuristic value η_e and an initial pheromone value τ_e is originally set to the inverse of the cost of traversing the edge e. τ_e is initially set to $\tau_0 = 1\backslash L_{nn}$ for all edge e, where L_{nn} is equal to the inverse of the tour length computed by the nearest-neighbor-heuristic algorithm.

Let q_0, α, $\rho \in [0,1]$, be real values and φ, β integer values between 0 and 8. For each vertex $s \in V$ a neighbor set is defined among the nearest vertices, N(s). For a given ant r, let NV(r) be the set of non-visited vertices. We denote $j_r(s) = N(s) \cap NV(r)$ the set of non-visited vertices among the neighbour set for a given vertex s and a given ant r.

In every iteration, each ant constructs a tour solution for the TSP-instance. The constructions phase works as follows:

Each ant is initially set in a randomly vertex, then at each step the entire ants make a movement to a non-visited vertex. Given an ant r with an actual position (vertex)s, p_{krs} is computed as a reference value for visiting or not vertex k, where:

$$p_{krs} = \begin{cases} \dfrac{[\tau(s,k)]^{\varphi}[\eta(s,k)]^{\beta}}{\sum_{u \in j(s)} [\tau(s,u)]^{\varphi}[\eta(s,u)]^{\beta}} & \text{if } k \in j(s) \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

This formula includes a small modification respect to the original ACS algorithm including φ as exponent of the pheromone level, this will allow us a deeper research on the effects of the possible combinations of, φ, β parameters.

A sample random value q is computed. If $q \le q_0$ we visit the city $k \in V$ with maximum p_{krs} (exploitation of the knowledge) otherwise ACS follows a random - proportional rule on p_{krs} for all $k \in V$ (biased exploration). If there are no non-visited vertex on the neighbor of vertex s, we extend 5 to all vertices in NV (r)/N (s) (those not visited by ant r and not included in the neighbor of s) and visits the vertex with maximum p_{krs}.

After an arc is inserted into a route (a new vertex is visited), its pheromone trail is updated. This phase is called Local update and for an inserted $e \in E$:

$$\tau_e = (1-\alpha)\tau_e + \alpha\tau_0 \quad (6)$$

This reduces the pheromone level in the visited arcs and the exploration in the set of possible tours is increased. When all the tours have been computed a global update phase is done. For each edge e pertaining to the global-best-tour found:

$$\tau_e = (1-\alpha)\tau_e + \alpha\Delta\tau_e \quad (7)$$

$$\Delta\tau_e = 1/L_{gb} \quad (8)$$

where, L_{gb} is the length of the global-best-tour found.

In the original ant algorithm and in most of the later versions, pheromone global update is performed in all the edges; ACS only updates pheromone level in the set of edges pertaining to the best tour.

We consider a trial as a performance of 1000 iterations. The lowest length tour found after all iterations are finished, is the best solution found by the trial. A feasible set of parameters for running ACS is a combination of feasible q_0, φ, β, α, ρ number of ants (na) and a concrete neighbor definition.

PSO-ACO ALGORITHM AND IMPLEMENTATION

The algorithm is run each time on a single TSP-instances. The set of parameters of ACS that define a point in the PSO domain are q_0, φ, β, ϕ, α, ρ and the number of ants (na). Most of them have already been explained. Let ϕ denotes the percentage of vertices that will be included in to N (v) for any vertex $v \in V$, so for a given $v \in V$ and $\phi = 0.5$ $|N(v)| = [\phi^*|V|]$. The ranges of each parameter are shown in Table 2, where each parameter pertains to its related] minimum, maximum]. DPSO =]0,1]x]-1,8]x]-1,8]x]0,1]x]0,1]x]0,1]x]0,40] $\subset \Re^7$. is the domain of the PSO. We define the fitness value of a given position (point) as the length of the best tour computed by an ACS using the related parameters in the given instance. If comparing two different positions they have the same length value then computing time is considered. We consider better of those parameters that minimize the length of the tour and secondly the time of computing. For computing the fitness of a given position, first integer parameters (na, φ and β) are rounded up as shown in Fig. 4, secondly the algorithm runs five trails of the ACS algorithm using the rounded parameters in the TSP-instance and returns the best value obtained from the trails.

For each particle of the population its initial velocity is set randomly. For half of the population the initial position is set using predefined parameters assuring that for every parameter there will be a

Table 2: Range of acs parameters q_0, φ, β, ρ, α, ϕ, na are the parameters used in ACS

	q_0	φ	β	ρ	α	ϕ	na
Minimum	0	-1	-1	0	0	0	0
Maximum	1	8	8	1	1	1	40

Point in the PSO domain (reflects a set of parameters)

(0.1 2.3 8 0.5 0.88 0.34 32.4)

Modified values to run on ACS:

(0.1 3 8 0.5 0.88 0.34 33)

Fig. 4: Modification of PSO points. In bold are the modified values.

particle containing a value covering the full range. The positions of the other half of the initial population is set randomly.

Parameters for the PSO have been set following[6,8] C1 = C2 = 2, χ = 0.729, the inertia weight is set initially to 1 and gradually decreasing from 1-0.1 (at each PSO iteration w = 0.99w). Maximum number of iterations is set to 500 due to computing time constraints (for 1000 PSO-iterations more than 1 day of computing time was necessary).

The algorithm PSO-ACS pseudo-code is as follows:

```
Select TSP-Instance.
Initialize particles.
Do 500
   For all the set of particles
      Position_Fitness PF = INFINITE
Do 5
   Perform a trial ACS with particle parameters.
   If New Value < PF
         PF = New value
     End if
   End Do
End for
Compute w = 0.99w
Update Best Parameters Found by each Particle
Update Best Parameters Found by the Population
Compute Velocity
Movement of Particles
   End Do
Return the set of parameters related to the best tour length found and the tour length.
```

The algorithm is based in a PSO framework, where particles are initialized and iteratively are moving though the domain of the set of parameters. The goal of the algorithm is, for a given instance to compute the tour with lowest length and to compute the set of ACS parameters, among those in DPSO, which gets the best ACS performance. Those final parameters are related with the TSP-instance selected.

RESULTS AND ANALYSIS

Algorithm was coded in C++. Algorithm has been run on six of the most widely used TSP-instances. Computational results are given in four parts: PSO-ACS behavior, PSO-ACS optimum values obtained, best set of parameters and comparison among sets of parameters performance.

Computationally, each PSO - ACS iteration shows a clear convergence: when the optimum (defined by the algorithm) number of ants and ϕ are nearly fixed, the

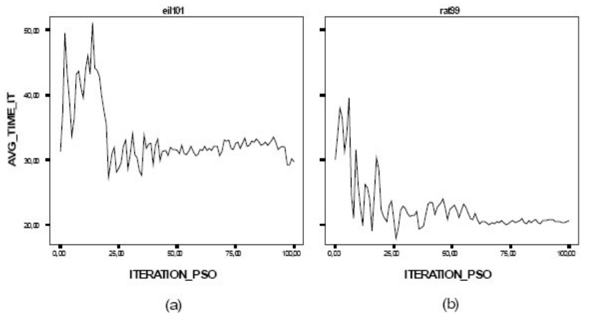

Fig. 5: Average time of the swarm at first 100 iterations for instances eil101 a and rat 99 b. AVG_TIME is average time of the iteration given a fixed number of particles and ITERATION_PSO is the number of the iteration in the PSO-ACS algorithm

Table 3: Sets of parameters

	α	β	ρ	φ	na	q_0	ϕ	Fitness
P_eil51	0.36	7	0.40	1	1	0.54	0.18	426
P_eil76	0.21	5	0.40	1	5	0.58	0.20	538
P_eil101	0.71	7	0.23	2	7	0.78	0.12	629
P_kroA100	0.64	4	0.24	1	4	0.64	0.12	21282
P_kroB100	0.71	1	0.08	1	3 9	0.86	0.12	22141
P_rat99	0.15	3	0.28	1	9	0.95	0.00	1211
ACS	0.10	2	0.10	1	1 0	0.9	a	b
ACS_GA	0.20	6	0.20	1	1 0	0.7	a	b

a have been tested for ϕ=0.1 0.2…..0.91 "b" there is no fitness value related. Values in bold mean optimum.

Computational time is also fixed (Fig. 5). In less than 100 iterations algorithm computes an optimum for integer parameters and in 200 iterations there are small differences among the optimum found and the particle's position for real parameters. In Fig. 6 we can see the evolution of the algorithm in the first 100 iterations. For the average of the fitness of the swarm(in a given iteration),there is a decreasing global tendency and after iteration 75 we can see the average of the fitness is kept on a fixed range, the size of this range is variable as shown in (c) and (d). For the minimum value obtained by the swarm in a given iteration, computational results show that at the beginning there are increasing and decreasing phases, when the particles are exploring their local optimums and moving also to the global one, but near iteration 100 the minimum is maintained as in (a) or frequently visited as in (b). This fast convergence can be an advantage as well a drawback because it can lead to a fast non-desirable convergence.

We set the reasons of this fast convergence in the PSO framework used and mainly in the method for evaluating a set of parameters: in a stochastic algorithm there is the probability that a bad set of parameters could perform well, if all the particles move into this

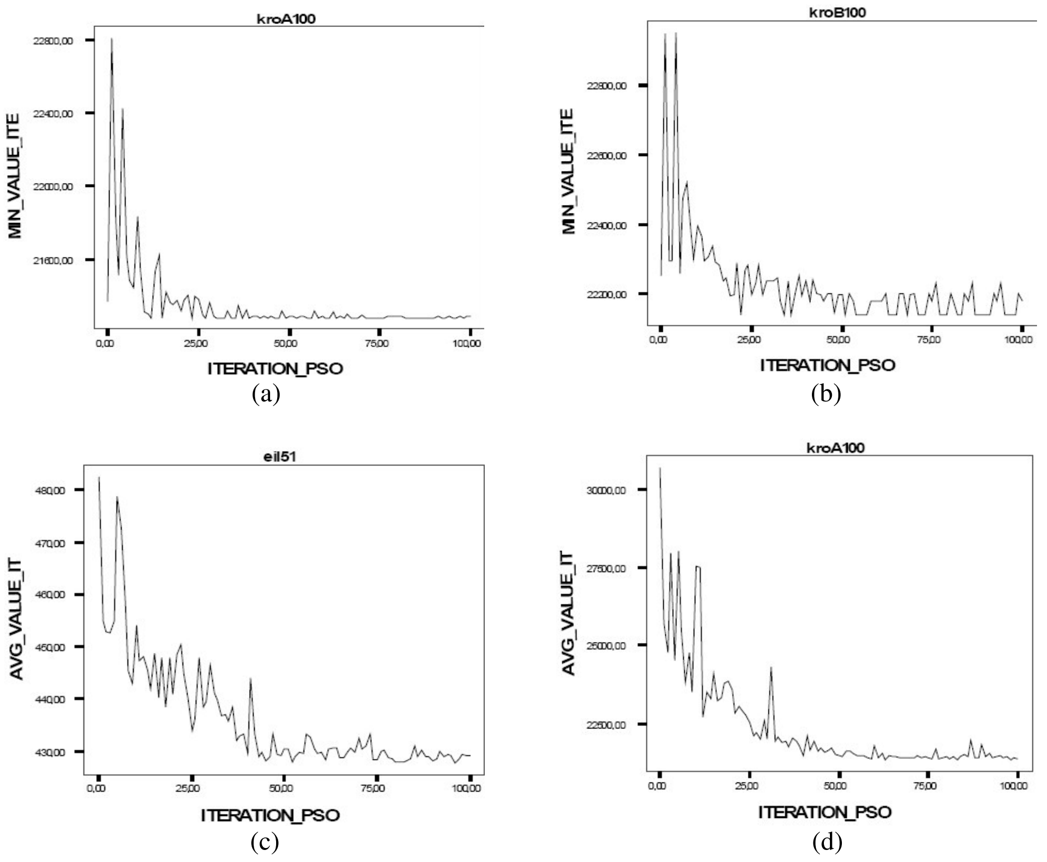

Fig. 6: First 100 iterations of the PSO-ACS algorithm. A and b are related to the minimum tour obtained at each iteration, c and d are related to the average of the particles fitness values. A and d are related to the instance kroA 100. b is related to kroB 100, and c to eil51. Those are the examples of typical behaviors in the 100 first iterations of the algorithm

area and the number of iterations in this area increases leading to probably good solutions that will cause the algorithm to remain in this non-optimal area.

Table 3 shows the optimum set of parameters found running PSO-ACS on each one of the instances selected.

CONCLUSION

In this research, a new approach is proposed for the convergence of associative neural memories by using the Fuzzy Particle Swarm Optimization technique (FPSO). The approach focuses on the neighborhood best and global best to increase the speed of convergence. In addition, this proposed model overcomes the local minima problem which is major drawback with the PSO technique.

The example illustrated suggests that our new approach can be used successfully as real time memory convergence technique for the artificial neural network.

Computational results seem to show that there is no uniquely optimal set of ACS parameters yielding best quality solutions in all the TSP instances. Nevertheless the PSO-ACS has been able to find a set of ACS parameters that work optimally for a majority of instances unlike others known so far.

PSO-ACS algorithm works well across different instances because it adapts itself to the instance characteristics. But it has a high computational overhead. A future work will try to modify the algorithm framework to reduce this cost.

PSO-ACS also has a fast convergence that can lead to a bad set of parameters. This may be due to two reasons: first is the specific PSO framework used and in modifying it we expect to obtain better results. Secondly the way the sets of parameters are evaluated may have to be reviewed as a bad set of parameters could lead to a non-desired convergence.

REFERENCES

1. Armij, L., 1996. Minimization of functions having Lischitz continuous first partial derivatives. Pacific J. Math., 16: 1-3.
2. Baladi and K. Hornik, 1989. Neural networks and principal component analysis: Learning from examples and local minima. Neural Networks, 2: 53-58.
3. Bello., M.G., 1992. Enhanced training algorithms and intigrated training/architecture selection for multi layer perceptron networks. IEEE Trans. Neural Networks, 3: 864-874.
4. Antsaklis, P.J and J.O. Moody, 1996. The dependence identification neural network construction algorithm. IEEE Trans. Neural Networks, 1: 3-15.
5. Baron Robert, H. and P. Coughlin James, 1994. Neural Computation in Hopfield Networks and Boltzmann Machines. Newyark: University of Delaware Press.
6. Kennedy, J. and R. C.Eberhart, 1995. Particle swarm optimization. Proceeding IEEE International Conference on Neural Networks. 4: 1942-1948.
7. Eberhart, R.C. and Y. Shi, 2001. Particle swarm optimization: Development, applications and resources. Proceeding Congress on Evolutionary Computation, IEEE Service Centre, Korea.
8. Dorigo, M., 1992. Optimization, Learning and Natural Algorithms. Ph.D Thesis. Dip Elettronica. Politecnico di Milano.
9. Dorigo, M., V. Maniezzo and A. Colorni, 1991. Positive Feedback as a Search Strategy. Technical Report. Dip Elettronica. Politecnico di Milano. pp: 91-116.
10. Dorigo, M., V. Maniezzo and A. Colorni, 1996. The Ant System: Optimization by a Colony of Cooperating Agents. IEEE Transaction on Systems, Man and Cybernetics. pp: 29-42.
11. Dorigo, M. and T. Stutzle, 2004. Ant Colony Optimization. MIT Press.
12. Dorigo, M. and L.M. Gambardella, 1997. Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem. IEEE Transaction on Evolutionary Computtin. pp: 53-66.
13. Gambardella, L.M and M. Dorigo, 1995. Ant-Q: A reinforcement Learning Approach to the Symmetric and Asymmetric Travelling Salesman Problems. Proceeding of the IEEE International Conference on Evolutionary Computation. pp: 252-260.
14. Pilat, E.C. and T. White, 2002. Using genetic algorithms to optimize ACS-TSP. Proceedings of the Third International Workshop on Ant Algorithms. pp: 282-287.
15. Kennedy, J. and R. Eberhart, 2001. Swarm Intelligence. Morgan Kaufmann Publishers.
16. Shi, Y. and R. Eberhart, 1998. Parameter selection in particle swarm optimization. IEEE International Conference on Evolutionary Computations. Lecture notes in Computer Science. Springer. pp:.591-600.
17. Shi, Y. and R. Eberhart, 1998. A modified particle swarm optimizer. IEEE International Conference on Evolutionary Computation. Anchorage, Alaska.
18. Mouzouris, G.C. and J.M. Mendel, 1996. Designing fuzzy logic systems for uncertain environments using a singular-value- QR decomposition method. Proceedings of the 5th IEEE International Conference on Fuzzy Systems, New Orleans. pp: 295-301.
19. Yen, J. and L. Wang, 1996. An SVD based fuzzy model reduction strategy. Proceeding of the 5th IEEE International Conference on Fuzzy System, New Orleans. pp: 835-841.
20. Yen, J and L. Wang, 1998. Granule-based Models. Handbook of fuzzy computation. IOP Publishing.
21. Yen, J. and L. Wang, 1999. Simplifying fuzzy rule-based models using orthogonal transformation methods. IEEE Transaction on Systems, man and Cybernatics. 29: 395-401.
22. Alter, O.O. Brown and D. Botstein, 2000. Singular value decompositin for genome-wide expression data processing and modeling. Proc. Natl Acad Sci USA.
23. Yen, J. and R. Langari, 2004. Fuzzy logic: Intelligence, control and information. Pearson Edition.