01 Jan 2004

Time Series Prediction with Recurrent Neural Networks using a Hybrid PSO-EA Algorithm

Nian Zhang

Ganesh K. Venayagamoorthy
Missouri University of Science and Technology

Donald C. Wunsch
Missouri University of Science and Technology, dwunsch@mst.edu

Xindi Cai

Recommended Citation

N. Zhang et al., "Time Series Prediction with Recurrent Neural Networks using a Hybrid PSO-EA Algorithm," *Proceedings of the IEEE International Joint Conference on Neural Networks, 2004*, Institute of Electrical and Electronics Engineers (IEEE), Jan 2004.
The definitive version is available at https://doi.org/10.1109/IJCNN.2004.1380208

Time Series Prediction with Recurrent Neural Networks Using a Hybrid PSO-EA Algorithm

Xindi Cai, Nian Zhang, Ganesh K.Venayagamoorthy and Donald C. Wunsch II

Applied Computational Intelligence Laboratory
Dept. of Electrical and Computer Engineering
University of Missouri - Rolla
Rolla, MO 65409, USA
E-mail: cai@umr.edu, nzhang@umr.edu, gkumar@ieee.org, dwunsch@ece.umr.edu

Abstract — **To predict the 100 missing values from the time series consisting of 5000 data given for the IJCNN 2004 time series prediction competition, we applied an architecture which automates the design of recurrent neural networks using a new evolutionary learning algorithm. This new evolutionary learning algorithm is based on a hybrid of particle swarm optimization (PSO) and evolutionary algorithm (EA). By combining the searching abilities of these two global optimization methods, the evolution of individuals is no longer restricted to be in the same generation, and better performed individuals may produce offspring to replace those with poor performance. The novel algorithm is then applied to the recurrent neural network for the time series prediction. The experimental results show that our approach gives good performance in predicting the missing values from the time series.**

I. INTRODUCTION

For the IJCNN 2004 time series prediction problem, an artificial time series with 5000 data is given, where 100 values are missing, as shown in Fig. 1. These missing values are divided in 5 blocks:

- elements 981 to 1,000;
- elements 1,981 to 2,000;
- elements 2,981 to 3,000;
- elements 3,981 to 4,000;
- elements 4,981 to 5,000.

This problem can be viewed as a prediction of the behavior of a system given previous observations of its input and output signals. Recurrent neural networks (RNN) have the capability to dynamically incorporate past experience due to internal recurrence. The feedback connections in their topologies, used to memorize past information, make them favorable in dealing with such temporal information processing problems [1].

The traditional gradient-based training algorithms, such as BPTT [2] and EKF [3], have been known to suffer from local minima and have heavy computation load for obtaining the derivative information. As the network architecture grows, the dimension of derivative matrices and approximate error covariance matrices in those algorithms increases exponentially, which makes them unfeasible for large scale recurrent network.

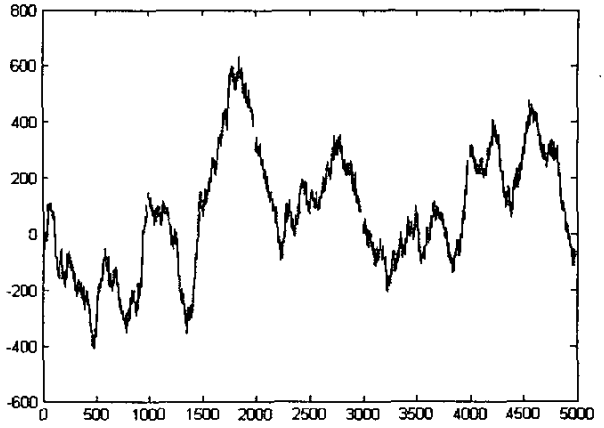

Fig. 1. Plot of the CATS time series consisting of 4900 given data points

Our approach employs a novel learning algorithm, which combines the particle swarm optimization (PSO) and evolutionary algorithm (EA). Evolutionary operators like selection and mutation have been integrated into the conventional PSO algorithm. By applying the selection operation in PSO, the particles with best performance are copied to next generation. Therefore, PSO can always keep the best performing particles [4]. The purpose of applying mutation operation to PSO is to increase the diversity of the population and thus overcome the local minima problem [5] in the PSO algorithm. The novel algorithm is then applied to train the recurrent neural network on the IJCNN 2004 time series.

The rest of the paper is organized as follows. Section II presents the architecture of the recurrent neural networks used. Section III describes the PSO, EA and hybrid PSO-EA learning algorithm. In Section IV, parameter selection is presented first, and then experimental results are provided. Finally, the conclusions are given in Section V.

0-7803-8359-1/04/$20.00 ©2004 IEEE

II. Recurrent Neural Networks

Elman's architecture was adopted in this paper which consists of a context layer, an input layer, two hidden layers, and an output layer, as shown in Fig. 2.

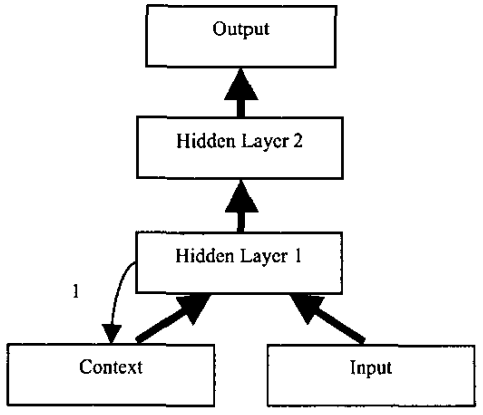

Fig. 2. Structure of the recurrent neural network

The context layer inputs in this system are obtained from the outputs of the first hidden layer units. The hidden unit activation pattern is copied verbatim through weight connections set equal to 1 and stored in the context units. The context layer has 40 neurons, the input layer has 100 neurons, the first hidden layer has 40 neurons, the second hidden layer has 20 neurons, and the output layer has 5 neurons. Neurons between adjacent layers are fully connected, as indicated by bold arrows in Fig. 2. The transfer functions of the two hidden layers and the output layer are *tansig*.

III. Hybrid PSO-EA Learning Algorithm

A. Particle Swarm Optimization

Particle swarm optimization (PSO) is a form of evolutionary computation technique developed by Kennedy and Eberhart [6], [7]. Similar to Evolutionary Algorithms (EA), particle swarm optimization algorithm is a population based optimization tool, where the system is initialized with a population of random solutions and the algorithm searches for optima satisfying some performance index over generations. It is unlike an EA, however, in that each potential solution is also assigned a randomized velocity, and the potential solutions, called *particles*, are then "flown" through the problem space.

Each particle has a position represented by a position vector $\vec{x}_i$. A swarm of particles moves through the problem space, with the velocity of each particle represented by a vector $\vec{v}_i$. At each time step, a function f_i representing a quality measure is calculated by using $\vec{x}_i$ as input. Each particle keeps track of its own best position, which is associated with the best fitness it has achieved so far in a vector $\vec{p}_i$. Furthermore, the best position among all the particles obtained so far in the population is kept track of as $\vec{p}_g$.

At each time step t, by using the individual best position, $\vec{p}_i(t)$, and global best position, $\vec{p}_g(t)$, a new velocity for particle i is updated by

$$\vec{v}_i(t+1) = w \times \vec{v}_i(t) + c_1\phi_1(\vec{p}_i(t) - \vec{x}_i(t)) + c_2\phi_2(\vec{p}_g(t) - \vec{x}_i(t)) \quad (1)$$

where c_1 and c_2 are positive constants, ϕ_1 and ϕ_2 are uniformly distributed random numbers in [0, 1] and w is the inertia weight. The term $\vec{v}_i$ is limited to the range $\pm \vec{v}_{max}$. If the velocity violates this limit, it is set at its proper limit. Changing velocity this way enables the particle i to search around its individual best position, $\vec{p}_i$, and global best position, $\vec{p}_g$. Based on the updated velocities, each particle changes its position according to the following:

$$\vec{x}_i(t+1) = \vec{x}_i(t) + \vec{v}_i(t+1) \quad (2)$$

Based on (1) and (2), the population of particles tends to cluster together with each particle moving in a random direction. The computation of PSO is easy and adds only a slight computation load when it is incorporated into EA. Furthermore, the flexibility of PSO to control the balance between local and global exploration of the problem space helps to overcome premature convergence of elite strategy in EA, and also enhances searching ability.

The pseudo code for PSO is summarized as follows [8]:

(1) Initialize a population of particles with random positions and velocities of d dimensions in the problem space.
(2) For each particle, evaluate the fitness according to the given fitness function in d variables.
(3) Compare current particle's fitness with its previous fitness. If current value is better than the previous, then set $\vec{p}_i$ value equal to the current value, and the $\vec{p}_i$ location equal to the current location in d-dimensional space.
(4) Compare fitness evaluation with the population's overall previous best position. If the current value is better than $\vec{p}_g$, then reset $\vec{p}_g$ to the current particle's array index and value.

(5) Change the velocity and position of the particle according to equation (1) and (2), respectively.
(6) Repeat step (2) to (6) until a criterion is met, usually a sufficiently good fitness or a maximum number of iterations/epochs.

B. Evolutionary Algorithm

To begin the evolutionary algorithm, a population of n neural networks, P_i, i=1,.., n, defined with weights and bias for each network, was created at random. Weights and biases were generated by sampling from a uniform distribution over [-1, 1]. Each neural network had an associated self-adaptive parameter vector σ_i, i=1,..., n, where each component corresponded to a weight or bias and served to control the step size of the search for new mutated parameters of the neural network. To be consistent with the range of initialization, the self-adaptive parameters for weights and biases were set initially to a uniform distribution over [-1, 1].

Each parent generated an offsprings strategy by varying all of the associated weights and biases. Specifically, for each parent P_i, i=1,..., n, an offspring P_i', i=1,..., n, was created by

$$\sigma_i'(j) = \sigma_i(j)\exp(\tau N_j(0,1)), \qquad j = 1,...,N_w \quad (3)$$

$$w_i'(j) = w_i(j) + \sigma_i' N_j(0,1), \qquad j = 1,...,N_w \quad (4)$$

where N_w is the number of weights and biases in the recurrent neural network, $\tau = 1/\sqrt{2\sqrt{N_w}}$, and $N_j(0,1)$ is a standard Gaussian random variable resampled for every j [9].

For the IJCNN 2004 time series prediction problem, a population of 40 individuals is competing for the best prediction. Each individual represents a RNN described in Section II. The number of weights and biases, i.e., N_w, in such a RNN (*size - 100x40x20x5*) is 6605, and hence τ is 0.0784. Half of population with best fitness is used as parents to create offspring for next generation.

C. Hybrid of PSO and EA

PSO works based on social and cognitive adaptation of knowledge, and all individuals are considered to be of same generation. On the contrary, EA works based on evolution from generation to generation, so the changes of individuals in a single generation are not considered. EA discard valuable information at the end of generation and starts almost randomly at next generation, while PSO keeps such information with the memory of local and global best throughout the entire evolution. On the hand, the property of mutation in EA helps to maintain the diversity of PSO population in "flying" to the new search area.

Based on the complementary property of PSO and EA, a new hybrid algorithm is created that combines the concepts of both algorithms. In each generation, the winners, which constitute half of the population, are enhanced by PSO. These winners are sharing information with each other as well as benefiting from their learning history, compared to EA where they are stagnant. The other half of the population which consists of individuals with lower fitness is replaced by the offspring created from the EA process with influence from the PSO enhanced parents. This procedure enhances the entire population. Fig. 3 illustrates this hybrid PSO-EA method.

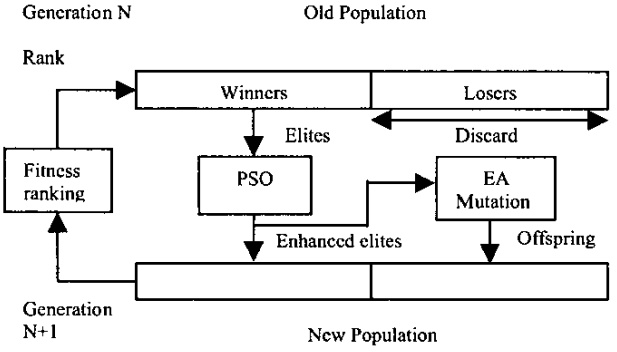

Fig. 3. Flow of hybrid PSO-EA method. Winners are enhanced by PSO and kept in the population for the new population. Offspring created by the enhanced elites of PSO as EA parents replaced the discarded losers in the old population generation to generation.

IV. RESULTS

A. PSO Parameters

The velocity change of a PSO, i.e. equation (1), consists of three parts. The first part is the momentum part, which prevents velocity to be changed abruptly. The second part is the "cognitive" part which represents private thinking of itself – learning from its own flying experience. The third part is the "social" part which represents the collaboration among particles – learning from group best's flying experience. The balance among these three parts determines the balance of the global and local search ability, therefore the performance of a PSO.

The inertia weight w controls the balance of global and local search ability. A large w facilitates the global search while a small one enhances local search. The introduction of an inertial weight also frees the selection of maximum velocity $\vec{v}_{max}$. The $\vec{v}_{max}$ can be reduced to $\vec{x}_{max}$, the dynamic range of each variable which is easier to learn; and the PSO performance is as good or better [10], [11].

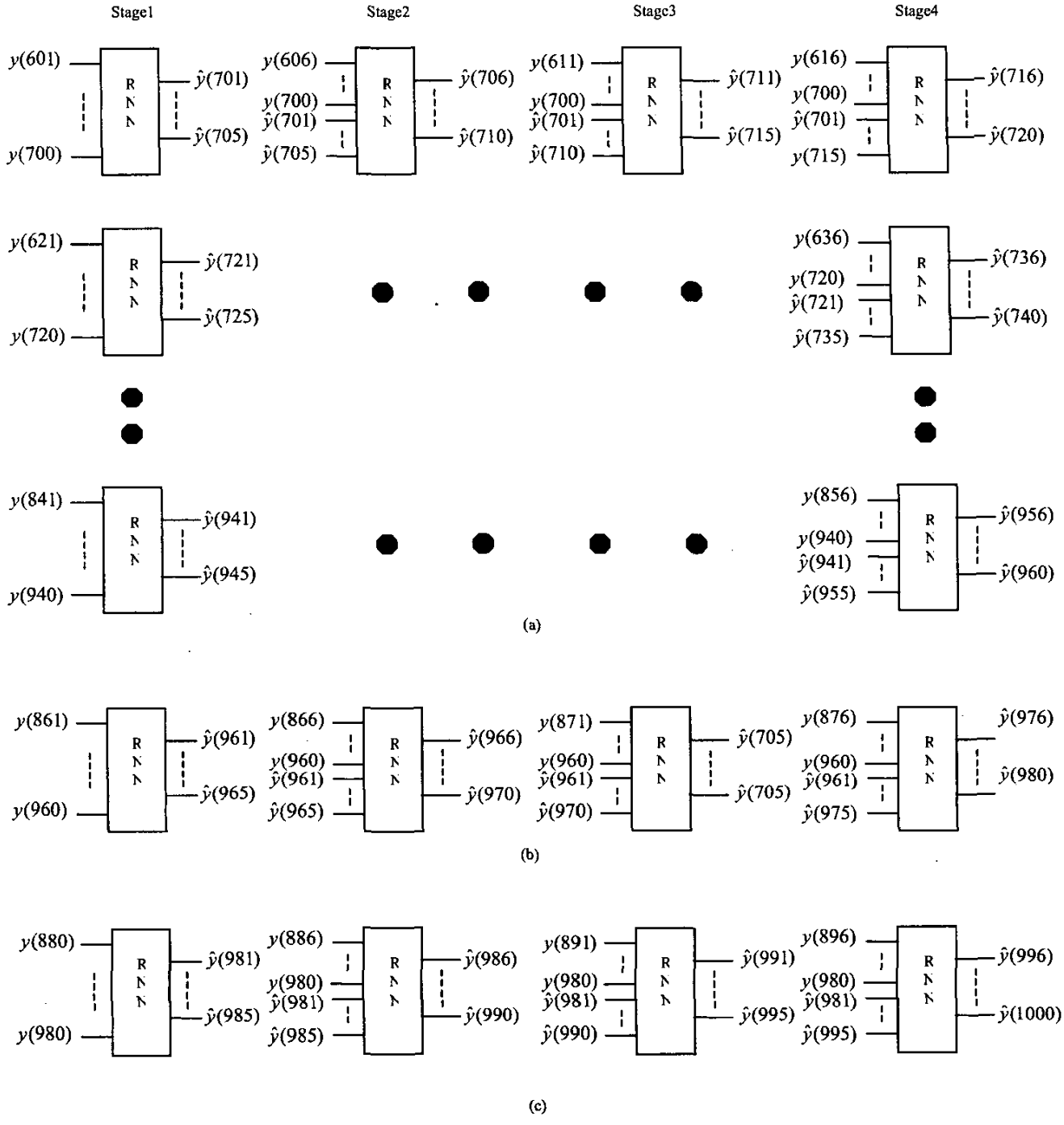

Fig. 4. The flow of RNN training, testing and predicting. (a) The RNN training procedure on one data set. (b) The RNN testing procedure on the same data set. (c) The real prediction for the 20 missing samples. y(n) denotes the original sample. $\hat{y}(n)$ denotes the RNN prediction.

Since the search process of a PSO is nonlinear and complicated, static parameters set, if well selected, can do a good job, but much better performance can be achieved if a dynamically changing scheme for the parameters is well designed, either a linearly decreasing inertia weight [10], a nonlinearly fuzzy changing [12], or involving a random component rather than time-decreasing [13]. All intuitively assume that the PSO should favor global search ability at the beginning and local search at the end.

Based on previous work [8], the authors have chosen the following parameters are chosen for the time series prediction:

Maximum velocity, V_{max}	*2*
Inertia weight, W	*0.8*
Acceleration constants, c_1, c_2	*2, 2*
Size of swarm	*40*

B. Training

We employ 5 identical RNNs (size: 100x40x20x5) for prediction on 5 data sets each containing 980 samples. The input vector is composed of both original samples and the network's previous predictions. At the first round, 100 consecutive original training samples are fed to the network to predict the next 5 points. When the prediction is available, the first 5 data points in the input vector are discarded, and network outputs, i.e. the predictions, are concatenated to the end of the input vector. This queue–like operation is continued until we have 20 predictions. The next round of training starts with another 100 consecutive original samples, whose starting point is 20 samples away from that of the previous 100, supplied to the network. The last 120 samples, 100 as inputs and 20 as targets, are allocated for testing purpose. When the network is well trained and tested, the last 100 samples are presented to predict the missing samples (see Fig. 4).

We use batch training method, and the weights are updated based on a cumulative error function. The process can be repeated over a number of epochs.

For the convenience, the original data are normalized on [-1, 1].

C. Results

Due to the assumption that missing data are only affected by recent samples, say nearest 200 ones, only a portion, 601^{st} - 940^{th}, of 980 samples in each set is used for training. The corresponding predictions on the training set are 701^{st} - 960^{th}. The last 100 samples, i.e. $861^{st} - 960^{th}$, are set side for testing network prediction on $961^{st} - 980^{th}$. The cumulative RMSE for the best individual drops to the range between magnitude order of 0.1 and 1 on original data after 2000 – 3000 generations. Fig. 5 shows the training errors on one data set, i.e. $4601^{st} - 4860^{th}$, for both the standard PSO and the hybrid PSO+EA. Fig. 6 illustrates the predictions on the above training data set by the hybrid PSO+EA. The predictions can be further improved by optimizing the PSO parameters as explained in [14].

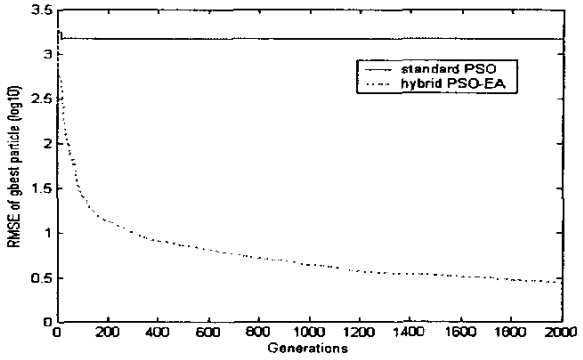

Fig. 5. Training error for the hybrid PSO-EA and the standard PSO. The errors reflect the performance of the best particle, i.e. the $\vec{p}_g$, at each generation for one data set.

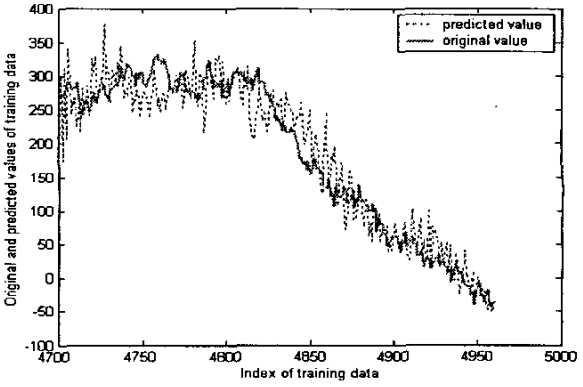

Fig. 6. Hybrid PSO+EA predictions by the gbest particle after 2000 generations.

V. CONCLUSIONS

We have presented and discussed a novel algorithm, hybrid PSO-EA. We explored how it combines the search capabilities of these two global optimization methods. The purpose of applying mutation in EA to PSO is to increase the diversity of the population and the ability to have the PSO to escape the local minima.

The hybrid PSO-EA learning algorithm proved to be successful in training RNN for the time series prediction. The hybrid procedure takes advantage of the complementary properties of PSO and EA, which makes it more powerful than each of the individual algorithms in enhancing the elites of a given population and generating offspring.

The results show that our approach gives a good prediction of the missing values from the given time series.

ACKNOWLEDGEMENT

The authors gratefully acknowledge the support from the National Science Foundation, the M. K. Finley Missouri endowment and the University of Missouri-Research Board.

REFERENCES

[1] Emad W. Saad, Danil V. Prokhorov, Donald C. Wunsch II, "Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks," *IEEE Transactions on Neural Networks*, vol. 9(6), pp. 1456 -1470, Nov. 1998.

[2] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of IEEE*, vol. 78, no. 10, pp. 1550-1560, Oct. 1990.

[3] G. V. Puskorius, and L. A. Feldkamp, "Neurocontrol of nonlinear dynamical systems with kalman filter trained recurrent networks," *IEEE Trans. Neural Networks*, Vol. 5 No. 2, pp. 279 -297, March 1994.

[4] P. J. Angeline, "Using selection to improve particle swarm optimization," *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 84-89, May 4-9, 1998.

[5] T. Blackwell and P. Bentley, "Don't push me! Collision-avoiding swarms," *Proceedings of the 2002 Congress on Evolutionary Computation*, vol. 2, pp. 1691-1696, May 12-17, 2002.

[6] James Kennedy and Russell Eberhart, "Particle Swarm Optimization," *IEEE International Conference on Neural Networks*, Perth, Australia, vol. 4, pp. 1942 -1948, Nov. 27-Dec. 1, 1995.

[7] Russell Eberhart and Yuhui Shi, "Particle swarm optimization: developments, applications and resources," *Proceedings of the 2001 Congress on Evolutionary Computation*, vol. 1, pp. 81 -86, May 27-30, 2001.

[8] V. G. Gudise, G. K. Venayagamoorthy, "Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks", *Proceedings of the IEEE Swarm Intelligence Symposium*, April 24- 26, 2003, Indianapolis, Indiana, USA, pp. 110 - 117.

[9] Kumar Chellapilla and David B. Fogel, "Evolution, neural networks, games, and intelligence," *Proceedings of the IEEE, Special Issue on Computational Intelligence*, vol. 87 (9), pp. 1471-1496.

[10] Y. Shi and R. C. Eberhart, "Parameter selection in particle swarm optimization," *Proceedings of the 1998 Annual Conference on Evolutionary Computation*, March 1998.

[11] Y. Shi and R. C. Eberhart, "A modified particle swarm optimizer," *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, pp. 69-73, May 1998.

[12] Y. Shi and R. C. Eberhart, "Fuzzy adaptive particle swarm optimization," *Proceedings of the 2001 Congress on Evolutionary Computation*, Seoul, Korea, 2001.

[13] Y. Shi and R. C. Eberhart, "Particle swarm optimization with fuzzy adaptive inertia weight," *Proceedings of the Workshop on Particle Swarm Optimization*, Indianapolis, IN, 2001.

[14] S Doctor, G K Venayagamoorthy, V G Gudise, "Optimal PSO for Collective Robotic Search Applications", *IEEE Congress on Evolutionary Computation*, June 19 – 23, 2004, Portland, OR, USA.