A Parallel Ant Colony Optimization Algorithm Based on Crossover Operation

Adem Kalinli[1] and Fatih Sarikoc[2]

[1] Department of Computer Technologies, Erciyes University, Kayseri Vocational High School, Kayseri, Turkey.
kalinlia@erciyes.edu.tr

[2] Department of Computer Engineering, Erciyes University, Institute of Science and Technology, Kayseri, Turkey.
fsarikoc@yahoo.com

Abstract

In this work, we introduce a new parallel ant colony optimization algorithm based on an ant metaphor and the crossover operator from genetic algorithms. The performance of the proposed model is evaluated using well-known numerical test problems and then it is applied to train recurrent neural networks to identify linear and nonlinear dynamic plants. The simulation results are compared with results using other algorithms.

Key words: Parallel Ant Colony Optimization, Hybrid Algorithms, Continuous Optimization, Recurrent Neural Network, System Identification

1 Introduction

There are many combinatorial optimization problems of the NP-hard type, and they cannot be solved by deterministic methods within a reasonable amount of time. The great difficulty of optimization problems encountered in practical areas such as production, control, communication and transportation has motivated researchers to develop new powerful algorithms. Therefore, several heuristics have been employed to find acceptable solutions for difficult real-world problems. The most popular of these new algorithms include genetic algorithms (GAs), simulated annealing (SA), ant colony optimization (ACO), tabu search (TS), artificial immune system (AIS), and artificial neural networks (ANNs) [1–4]. Although all of these algorithms convergence to a global optimum, they cannot always guarantee optimum solutions to the problem. Therefore, they are called approximate or heuristic algorithms.

The ACO algorithm is an artificial version of the natural optimization process carried out by real ant colonies. The first ACO algorithm was proposed by Dorigo et al. in 1991, and was called an ant system (AS) [5, 6]. Real ants communicate with

each other by leaving a pheromone substance in their path, and this chemical substance leads other ants. Thus, stimergy is provided and swarm intelligence emerges in the colony behaviour. The main features of the algorithm are distributed computation, positive feedback and constructive greedy search. Since 1991, several studies have been carried out on new models of the ACO algorithm and their application to difficult optimization problems. Some of these algorithms are known as AS with elitist strategy (AS_{elit}), rank based version of AS (AS_{rank}), MAX-MIN AS and ant colony system (ACS) [7–10]. In most application areas, these algorithms are mainly used for optimization in discrete space [9–13]. In addition, different kinds of ant algorithms, such as continuous ant colony optimization (CACO), API, continuous interacting ant colony (CIAC) and touring ant colony optimization (TACO), have been introduced for optimization in the continuous field [14–17].

It is known that there is a premature convergence (stagnation) problem in the nature of ant algorithms [6]. Therefore, as the problem size grows, the ability of the algorithm to discover the optimum solution becomes weaker. On the other hand, when the problem size and number of parameters increase, parallel implementation of the algorithm could give more successful results [18,19]. Furthermore, ant colony optimization approaches are population based and they are naturally suited to parallel implementation [18–22]. So, these advantages lead us to consider a parallel version of the ant algorithm.

In this work a parallel ant colony optimization (PACO) algorithm based on the ant metaphor and the crossover operator of GAs is described. Our aim is to avoid premature convergence behaviour of the ant algorithm and to benefit from the advantages of a parallel structure. The performance of the proposed PACO algorithm is compared to that of the basic TS, parallel TS (PTS), GA and TACO algorithms for several well-known numerical test problems. Then, it is employed to train a recurrent neural network to identify linear and nonlinear dynamic plants. The second section of the chapter presents information about parallel ant colony algorithms in the literature. In the third section, the basic principles of TACO algorithms are introduced and the proposed model is described. Simulation results obtained from the test functions optimization and an application of PACO to training recurrent neural network are given in the fourth section. The work is concluded in the fifth section.

2 Parallel Ant Colony Algorithms

There are a few parallel implementations of ant algorithms in the literature. The first of these studies is that of Bolondi and Bondanza. They used fine-grained parallelism and assigned each ant to a single processor. Due to the high overhead for communication, this approach did not increase performance with an increased number of processors. Better results have been obtained with a more course-grained model [20,23].

Bullnheimer et al. propose two parallelization strategies, synchronous and partially asynchronous implementations of the ant system [18]. In simulations made on some TSP instances, it is shown that the synchronization and communication overhead slows down the performance. For this reason, the asynchronous parallel version outperforms the synchronous version as it is reduces the communication frequency.

Stützle, using some TSP instances, empirically tests the simple strategy of executing parallel independent short runs of a MAX-MIN ant system [19]. He compares the solution quality of these short runs with the solution quality of the execution of one long run whose running time equals the sum of the running times of the short runs. He shows that using parallel independent runs with different initial solutions is very effective in comparison with a single long run.

Talbi et al. implemented a synchronous master–worker model for parallel ant colonies to solve the quadratic assignment problem [22]. At each iteration, the master broadcasts the pheromone matrix to all the workers. Each worker receives the pheromone matrix, constructs a complete solution by running an ant process, applies a tabu search for this solution as a local optimization method and sends the solution found to the master. According to all solutions, the master updates the pheromone matrix and the best solution found, and then the process is iterated.

Michel et al. propose an island model approach inspired by GAs [24]. In this approach every processor holds a colony of ants and in a fixed number of generations each colony sends its best solution to another colony. If the received new solution is better, then it becomes the new solution for the colony and pheromone updating is done locally depending on this new solution. Thus, the pheromone matrices of colonies may differ from each other.

Delisle et al. presented a shared memory parallel implementation of an ant colony optimization for an industrial scheduling problem in an OpenMP environment [25].

In another implementation, Krüger et al. indicate that it is better to exchange only best solutions found so far than to exchange the whole pheromone matrix [26].

Middendorf et al. show that information exchanges between colonies in small quantities decrease the run time of the algorithm and improve the quality of the solutions in multi-colony ant systems. They also conclude that it is better to exchange local best solutions only with a neighbour in a directed ring and not too often, instead of exchanging the local best solution very often and between all colonies [20].

3 Touring Ant Colony Optimization and Proposed Parallel Model

3.1 Pheromone Based Feedback in Ant System Metaphor

Real ants are capable of finding the shortest path from their nest to a food source, back or around an object. Also, they have the ability to adapt to changes in the environment. Another interesting point is that ants are almost blind, in other words they cannot see well enough to select directions to follow. Studies on ants show that their ability to find the shortest path is the result of chemical communication among them. They use a chemical substance called pheromone to communicate with each other. This type of indirect interaction through modification of the environment, which is called stimergy, is the main idea of ACO algorithms.

Ants deposit a certain amount of pheromone on their path while walking and each ant probabilistically chooses a direction to follow. The probability degree of being the chosen direction depends on the pheromone amount deposited on that direction. If the pheromone amount of all directions is equal, then all directions have

the same probability of being preferred by ants. Since it is assumed that the speed of all ants is the same and, therefore, all ants deposit the same amount of pheromone on their paths, shorter paths will receive more pheromone per time unit. Consequently, large numbers of ants will rapidly choose the shorter paths. This positive feedback strategy is also called an auto-catalytic process. Furthermore, the quantity of pheromone on each path decreases over time because of evaporation. Therefore, longer paths lose their pheromone intensity and become less attractive as time passes. This is called a pheromone-based negative feedback strategy.

If there are only a few ants, the auto-catalytic process usually produces a bad-optimal path very quickly rather than an optimal one. Since there are many ants searching simultaneously for the optimum path, the interaction of these auto-catalytic processes causes the search to converge to the optimum path very quickly and to finally find the shortest path between the nest and the food without getting stuck in a sub-optimal path. The behaviour of real ant colonies when finding the shortest path represents a natural adaptive optimization process.

The ant colony optimization algorithm is an artificial version of the natural optimization process carried out by real ant colonies as described above. A simple schematic algorithm modeling the behaviour of real ant colonies can be summarized as below:

BEGIN
Initialize
REPEAT
  Generate the artificial paths for all ants
  Compute the length of all artificial paths
  Update the amount of pheromone on the artificial paths
  Keep the shortest artificial path found up to now
UNTIL (iteration = maxiteration or a criterion is satisfied)
END.

3.2 Touring Ant Colony Optimization Algorithm

In this algorithm, a solution is a vector of design parameters which are coded as a binary bit string. Therefore, artificial ants search for the value of each bit in the string. The concept of the TACO algorithm is shown in Fig. 1.

At the decision stage for the value of a bit, ants use only the pheromone information. Once an ant completes the decision process for the values of all bits in the string,

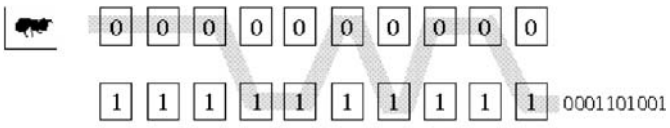

Fig. 1 An artifical path (solution) found by an ant

it means that it has produced a solution to the problem. This solution is evaluated in the problem and a numerical value showing its quality is assigned to the solution using a function, often called the fitness function. With respect to this value, an artificial pheromone amount is attached to the links, forming the artificial way, between the chosen bits. An ant on the nth bit position chooses the value of 0 or 1 for the bit on the $(n+1)$th position depending on the probability defined by the following equation:

$$p_{ij}(t) = \frac{[\tau_{ij}]^{\alpha}}{\sum_{j=1}^{2} [\tau_{ij}]^{\alpha}} \tag{1}$$

where $p_{ij}(t)$ is the probability associated with the link between bit i and j, $\tau_{ij}(t)$ is the artificial pheromone of the link, α is a weight parameter. Artificial pheromone is computed by the following formula:

$$\Delta\tau_{ij}^{k}(t, t+1) = \begin{cases} \frac{Q}{F_k} & \text{if the ant } k \text{ passes the link } (i, j) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where $\Delta\tau_{ij}^{k}$ is the pheromone quantity attached to the link (i, j) by the artificial ant k, Q is a positive constant and F_k is the objective function value calculated using the solution found by the ant k.

After M ants complete the search process and produce their paths, the pheromone amount to be attached to the sub-path $(0 \rightarrow 1)$ between time t and $(t+1)$ is computed as

$$\Delta\tau_{ij}(t, t+1) = \sum_{k=1}^{M} \Delta\tau_{ij}^{k}(t, t+1) \tag{3}$$

The amount of pheromone on the sub-path (i, j) at the time $(t+1)$ is calculated using the following equation:

$$\tau_{ij}(t+1) = \rho\tau_{\mathrm{ij}}(t) + \Delta\tau_{ij}(t, t+1) \tag{4}$$

where ρ is a coefficient called the evaporation parameter.

3.3 Parallel Ant Colony Optimization Algorithm

In this work, we introduce a hybrid algorithm model to avoid premature convergence of ant behaviour and to obtain a robust algorithm. Generally, hybrid models utilize the benefits of different algorithms. The proposed parallel ant colony optimization (PACO) algorithm is based on the data structure of the TACO and the crossover operator of GAs. We combine the convergence capability of the ant metaphor and the global search capability of the genetic algorithm.

In the PACO algorithm, each solution is represented by a binary vector of design parameters and artificial ants search for the value of each bit in the string as TACO. For this reason, the proposed algorithm can search a sampled finite subset of continuous space.

The flowchart and pseudocode of the proposed model is given in Fig. 2 and Fig. 3, respectively. In the model, different independent ant colonies are executed in parallel. Each colony has a copy of the same search space with the same initial pheromone quantities. However, it is possible to use different control parameter values for each colony. As the algorithm runs, the pheromone quantities of each copy may be different. A colony does not change the pheromone quantities of another colony. However, they have the ability to exchange information implicitly. The information exchange process between the ant colonies is based on the crossover operation.

Execution of the colonies is stopped after a given number of iterations (*NumAntCycle*). There is no specific rule to determine this number; it may be defined experimentally. *NumAntCycle* is normally chosen to be sufficiently large to allow the search to complete local searching. When all ants complete their paths in a colony, the quality of the path produced by each ant is evaluated and then the best one found is reserved as the local best of the colony. In every fixed number of *NumAntCycle* iterations, local best solutions of each colony are added to the solution population. Later, this population is altered by a crossover procedure to produce a new population. This new population is formed by implementing the crossover operation among the solutions belonging to the previous solution population. After crossover, the best part of the population survives and the solutions of this part are used to update the pheromone quantities of the best paths in each colony. Thus, one epoch of the algorithm is completed. In successive epochs, the search continues, depending on the pheromone quantities updated in the previous epoch. This process is repeated until a predefined number of epochs (*NumOfEpoch*) is completed. *NumOfEpoch* may change according to the problem, so the value of this parameter is experimentally defined.

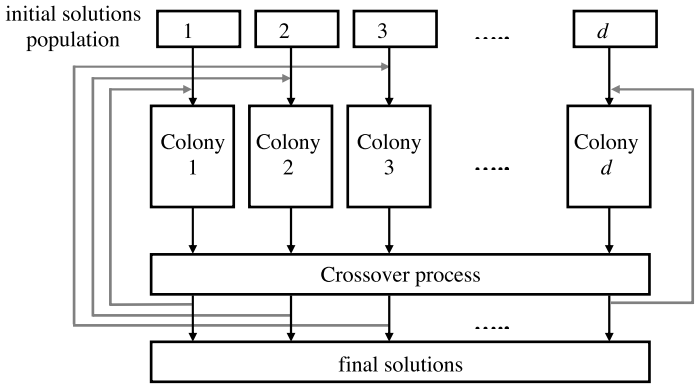

Fig. 2 Flowchart of the PACO algorithm

Step 1) /* initialization */
For *d*:=1 to *NumberOfColony* do /* pheromone initialization */
For *j*:=1 to *LenghtOfBitString* do
$\tau_d(0, j) = c$
$\tau_d(1, j) = c$
End
End
For *d*:=1 to *NumberOfColony* do /* initial population of solutions */
$BestSolBitStr_d^{old} = FuncRandFeasible()$
Step 2) /* movement of ants */
For *d*:=1 to *NumberOfColony* do
For *j*:=1 to *LenghtOfBitString* do
For *k*:=1 to *NumberOfAnts* do
Select a bit value 0 or 1 according to equation (5) and (6)
Assign chosen value to $tabu_d^k(j)$
chosen bit value $i = tabu_d^k(j)$
$\tau_d^{new}(i, j) = (1 - \rho).\tau_d^{old}(i, j) + \rho.\tau_0$ /* local update */
End
End
End
Step 3) /* Evaluate ant solutions */
For *d*:=1 to *NumberOfColony* do
For *k*:=1 to *NumberOfAnts* do
$CostOfSol_d^k = FuncCost\left(tabu_d^k(j)\right)$ /* *j*=1 to *LenghtOfBitString* */
Reserve best solution bit string of each colony as $BestSolBitStr_d^{new}$
End
End
Step 4) /* Empty all tabu list */
Step 5) /* Crossover operation */
If (*NumAntCycle* is completed) Then do /*crossover condition is provided*/
Mate $BestSolBitStr_d^{old}$, $BestSolBitStr_d^{new}$
Produce offspring $BestSolBitStr_d^{offspring}$
Evaluate cost of offspring solutions
Survive best part of the population
Keep the best solutions as $BestSolBitStr_d^{old}$ and the costs as $CostOfBestSol_d$
For *d*:=1 to *NumberOfColony* do /* crossover update */
For *j*:=1 to *LenghtOfBitString* do
$i = BestSolBitStr_d^{old}(j)$
$\tau_d^{new}(i, j) = (1 - \rho).\tau_d^{old}(i, j) + 1/\left(a + \left|CostOfBestSol_d\right|\right)$
End
End
End-if
Step 6) If (*NumOfEpoch* is completed) Then Stop /* Stopping criteria is satisfied */
Else Goto Step 2

Fig. 3 Pseudocode of the proposed algorithm

Different independent ant colonies are sequentially executed in a single processor, for this reason, implementation of the algorithm is virtually parallel. Communication between the colonies is carried out at predetermined moments; therefore the parallelism used in this work is synchronous.

Crossover Procedure

The crossover operator employed by GAs is used to create two new solutions (children) from two existing solutions (parents) in the population. Depending on the method of problem representation in string form, a proper crossover operator must be chosen. When the problem is represented in binary string form, the simplest crossover operation can be applied as follows: two solutions are randomly selected as parent solutions from the population at two randomly selected points. The parts between the points are swapped and two new solutions are produced. A crossover operation can thus yield better solutions by combining the good features of parent solutions. An example of this simple crossover operator is given below:

PresentSolution1 **101 1101 01110**
PresentSolution2 110 0011 11011

NewSolution1 **101** 0011 **01110**
NewSolution2 110 **1101** 11011

Movements of Ants

In the proposed model, the data representation structure is defined as discrete elements in a matrix form. Rows of this matrix are indicated by the values of i and columns are indicated by the values of j. Since binary data representation is used, i can take the values 0 or 1, but the maximum value of j depends on the parameters of the problem. The element (i, j) of a predefined matrix format addresses a point in the data structure on which artificial ants move as depicted in Fig. 1.

At the beginning of the search, some initial pheromone quantity (c) is allocated to each path of the binary coded search space. In addition, an initial population of random solutions in the feasible region is formed for genetic crossover operations on the succeeding population.

While ants move from one point to another, they search the value of each bit in the string, in other words, they try to decide whether the value of the next bit to be chosen is 0 or 1 according to the state transition rule given in Equation (5).

$$\text{tabu}_d^k(j) = \begin{cases} \text{argmax}_{i\in\{0,1\}} \{\tau_d(i,j)\} & \text{with the probability of } q_0 \text{ (exploit)} \\ S_d(j) & \text{with the probability of } (1-q_0) \text{ (explore)} \end{cases} \tag{5}$$

where $\text{tabu}_d^k(j)$ means the jth element of the tabu list for the kth ant in the dth colony. In other words, it represents the selected value for the next bit. $\tau_d(i,j)$ is accumulated pheromone substance on the path (i, j) belonging to the dth colony

and $S_d(j)$ is a stochastically found new value of the jth element for the dth colony ($S_d(j) \in \{0, 1\}$). In this equation, ants take a deterministic decision with the probability q_0, which is an initial coefficient balancing deterministic search versus stochastic search. Deterministic search exhausts accumulated pheromone knowledge to find new solutions near to the best one found so far. On the other hand, stochastic search explores possible new paths to enhance the searching area. $S_d(j)$ is defined according to the probability distribution formula

$$P_d(i,j) = \frac{\tau_d(i,j)}{\tau_d(j,0) + \tau_d(j,1)}, \quad i \in \{0, 1\} \tag{6}$$

where $P_d(i,j)$ represents probability as an indication of accumulated pheromone attraction of the point (i,j), which is to be selected in the search space of the dth colony. To select a bit value $S_d(j)$ for the jth element of the tabu list, a stochastic decision mechanism can be implemented over the probability distribution formula $P_d(i,j)$. With this formula a roulette wheel mechanism could be used as a decision mechanism in order to define the next value of $S_d(j)$.

After choosing a path each ant updates the pheromone level of the path. This operation is called the *local updating rule* and the formula for the rule is

$$\tau_d^{new}(i,j) = (1-\rho).\tau_d^{old}(i,j) + \rho.\tau_0 \tag{7}$$

This formula is implemented in order to make the way previously chosen less attractive and to direct the following ants to other paths. ρ is an initial coefficient representing evaporation rate and τ_0 is another coefficient showing the minimum level of pheromone instances in each path.

After a predefined number of cycles (*NumAntCycle*), each colony reserves its best solution found so far and this solution is added to the solution population for the crossover operation. The crossover operation eliminates the worse part of the population and provides a global reinforcement mechanism. Each solution of the surviving part is assigned to one of the colonies, thus, information exchange between colonies is implicitly provided. After this process, pheromone values of each colony are updated depending on the returned solutions according to the formula

$$\tau_d^{new}(i,j) = (1-\rho).\tau_d^{old}(i,j) + \frac{1}{(a + |\text{cost\_d}|)} \tag{8}$$

where cost_d is a value calculated by the cost function related to the assigned solution to the dth colony. a is a constant employed to avoid overflow and scale the pheromone effect of the cost value. Determining a proper value for a is highly dependent on the problem. So, some preliminary experience and knowledge about the range of the cost function values is necessary. This may be considered as a weakness of the algorithm. Furthermore, by employing a crossover procedure, the proposed algorithm moves far from a realistic simulation of ants in order to increase the search capability of the global optimum and to yield better performance.

It is known that there are reports in the literature based on the concepts TACO and Island Model. The proposed algorithm uses a binary data representation, which was earlier used in the TACO algorithm [17]. However, PACO differs from TACO in some features. First, in PACO, selection of the next point depends on a *state transition rule* instead of using only a probability distribution formula. By employing a state transition rule, PACO is able to balance exploitation versus exploration with a certain probability as happened in ACS. Second, after selection of each new point, a *local updating formula* is implemented in order to lead other ants to unselected paths. Moreover, crossover procedure is employed at predetermined intervals to provide information exchange between colonies. Thus, PACO uses synchronous and parallel information exchange structures established in a multi-colony ant system. Island Model is another hybrid algorithm that combines both ACO and GA. However, this model has a different data representation structure and runs more than one processor [24]. Since each processor holds a colony, implementation of the algorithm is more complicated than PACO and TACO.

4 Simulation Results

The simulation work consists of two parts: numeric function optimization and training an Elman network to identify dynamical linear and non-linear systems.

4.1 Continuous Function Optimization

Seven well-known minimization test functions were employed to determine the performance of the proposed PACO algorithm. These test functions are given in Table 1.

Table 1 Numerical test functions used in the simulations

Notation	Name	Function
F1	Sphere	$f_1 = \sum_{i=1}^{4} x_i^2$
F2	Rosenbrock	$f_2 = 100\left(x_1^2 - x_2\right)^2 + \left(1 - x_1\right)^2$
F3	Step	$f_3 = \sum_{i=1}^{5} [x_i]$, where $[x_i]$ represents the greatest integer less than or equal to x_i
F4	Foxholes	$f_4 = \left[0.002 + \sum_{j=1}^{25} \left(j + \sum_{i=1}^{2} \left(x_i - a_{ij}\right)^6\right)^{-1}\right]^{-1}$ $\{(a_{1j}, a_{2j})\}_{j=1}^{25}$ = (-32,-32), (-16,-32), (0,-32), (16,-32), (32,-32), (-32,-16), (-16,-16), (0,-16), (16,-16), (32,-16),..., (-32,32), (-16,32), (0,32), (16,32), (32,32)
F5		$f_5 = \left(x_1^2 + x_2^2\right)/2 - \cos\left(20\pi x_1\right)\cos\left(20\pi x_2\right) + 2$
F6	Griewangk	$f_6 = 1 + \sum_{i=1}^{10} \left(\frac{x_i^2}{4000}\right) - \prod_{i=1}^{10} \left(\cos\left(\frac{x_i}{\sqrt{i}}\right)\right)$
F7	Rastrigin	$f_7 = 20A + \sum_{i=1}^{20} \left(x_i^2 - 10\cos\left(2\pi x_i\right)\right), A = 10$

The first four test functions were proposed by De Jong [27]. All test functions reflect different degrees of complexity.

Sphere (F1) is smooth, unimodal, strongly convex, and symmetric.

Rosenbrock (F2) is considered to be difficult, because it has a very narrow ridge. The tip of the ridge is very sharp, and it runs around a parabola.

Step (F3) is a representative of the problems of flat surfaces. Flat surfaces are obstacles for optimization algorithms because they do not give any information as to which direction is favourable. The background idea of the step function is to make the search more difficult by introducing small plateaus to the topology of an underlying continuous function.

Foxholes (F4) is an example of a function with many local optima. Many standard optimization algorithms get stuck in the first peak they find.

Function F5 has 40000 local minimum points in the region when x_1 and x_2 are within $[-10, 10]$.

Griewangk (F6) is also a non-linear and multi-modal function. The terms of the summation produce a parabola, while the local optima are above parabola level. The dimensions of the search range increase on the basis of the product.

Rastrigin's function (F7) is a fairly difficult problem due to the large search space and large number of local minima. This function contains millions of local optima in the interval considered.

The solutions for the functions, parameter bounds, resolutions and the length of each solution for each test function are given in Table 2.

In the first stage, simulation results were obtained for the proposed model. The proposed PACO was executed 30 times with different initial solutions. The number of ant colonies running in parallel was 4 (*NumOfCol*) and the number of ants was 30 (*NumOfAnts*). Each colony at any epoch was run for 20 (*NumAntCycle*) iterations for the first five functions and 50 for the others. The total number of cycles made at any epoch was 80 ($NumOfCol \cdot NumAntCycle$) for the first five functions and 200 for the other two functions. This process was repeated through 16800 evaluations for the first five functions, 50000 evaluations for the other two functions in order to compare the performance of the proposed method with the results obtained using GA, TS, PTS,

Table 2 Number of parameters, solutions, parameter bounds and length of solution for the test functions

Function	Number of parameters	Solutions		Parameter bounds		Length of a solution
		x_i	$f(x)$	Lower	Upper	
F1	4	0.0	0.0	−5.12	5.12	40
F2	2	1.0	0.0	−2.048	2.048	32
F3	5	−5.12	−30.0	−5.12	5.12	50
F4	2	−32.0	1.0	−65536	65536	40
F5	2	0.0	1.0	−10	10	36
F6	10	0.0	0.0	−600	600	200
F7	20	0.0	0.0	−5.12	5.12	400

and TACO algorithms taken from [4] and after that the search was stopped. Other parameters of the PACO were chosen as $c = 0.1$, $\rho = 0.1$, $\tau_0 = 0.1$, $q_0 = 0.9$, $a = 0.001$.

To show the robustness of the proposed model, frequency histograms of the results obtained using GA, TACO, basic TS, PTS and PACO algorithms are given in Figs. 4–10 for the test functions 1–7, respectively.

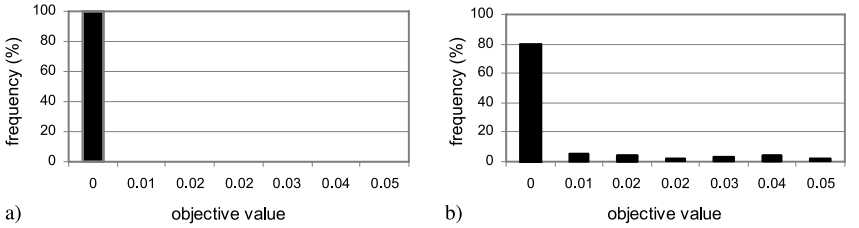

Fig. 4 Histograms drawn from the results obtained for the function F1 by (**a**) TS, PTS, TACO and PACO algorithms, (**b**) GA

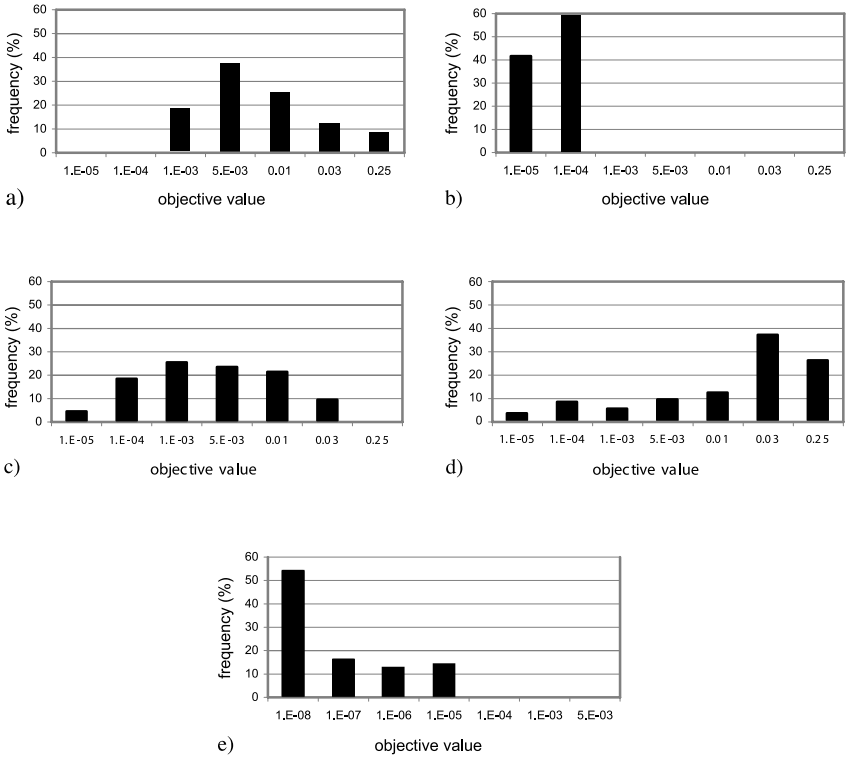

Fig. 5 Histograms drawn from the results obtained for the function F2 by (**a**) TS, (**b**) PTS, (**c**) GA, (**d**) TACO, and (**e**) PACO

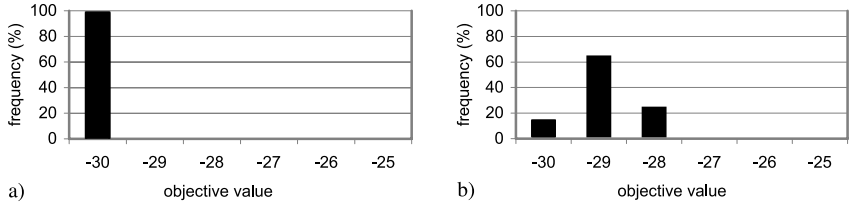

Fig. 6 Histograms drawn from the results obtained for the function F3 by (**a**) TS, PTS, GA, and PACO, (**b**) TACO

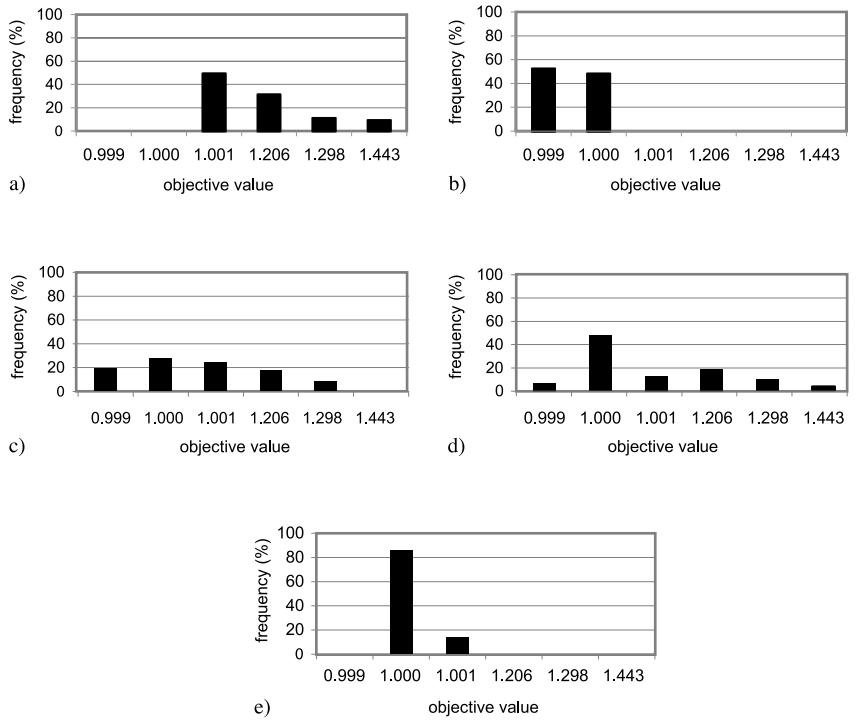

Fig. 7 Histograms drawn from the results obtained for the function F4 by (**a**) TS, (**b**) PTS, (**c**) GA, (**d**) TACO, and (**e**) PACO

4.2 Training Recurrent Neural Network by Using the PACO Algorithm

The use of artificial neural networks (ANNs) to identify or model dynamic inputs is a topic of much research interest. The advantage of neural networks for these types of applications is to learn the behaviour of a plant without much *a priori* knowledge about it. From a structural point of view, there are two main types of neural networks: feedforward neural networks (FNNs) and recurrent neural networks (RNNs) [28]. Connections that allow information to loop back to the same processing element are called recursive and NNs having these types of connections are named RNNs.

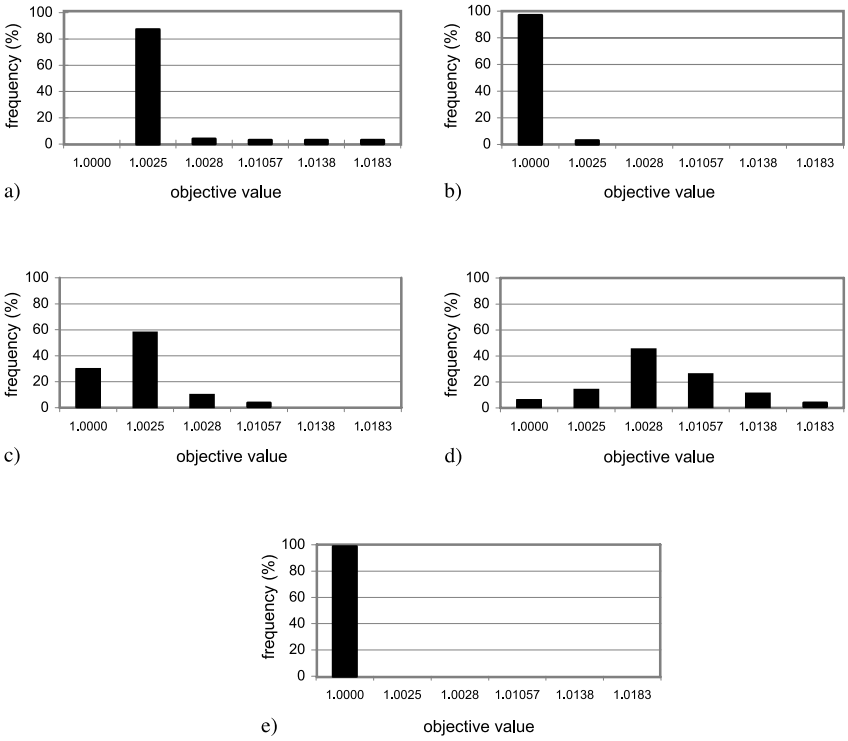

Fig. 8 Histograms drawn from the results obtained for the function F5 by (**a**) TS, (**b**) PTS, (**c**) GA, (**d**) TACO, and (**e**) PACO

RNNs are more suitable than FNNs for representing a dynamic system since they have a dynamic mapping between their output(s) and input(s). RNNs generally require less neurons in the neural structure and less computation time. Moreover they have a low probability of being affected by external noise. Because of these features, RNNs have attracted the attention of researchers in the field of dynamic system identification.

Although gradient based search techniques such as back-propagation (BP) are currently the most widely used optimization techniques for training neural networks, it has been shown that these techniques are severely limited in their ability to find global solutions. Global search techniques such as GA, SA, TS and ACO have been identified as a potential solution to this problem. Although the use of GAs for ANN training has mainly focused on FNNs [29–31], there are several works on training RNNs using GAs in the literature [32–34]. SA and TS have some applications for the training of ANNs [35–37]. Although GA, SA and TS algorithms have been used for training some kinds of neural networks, there are few reports of use of the ACO algorithm to train neural networks [38–40].

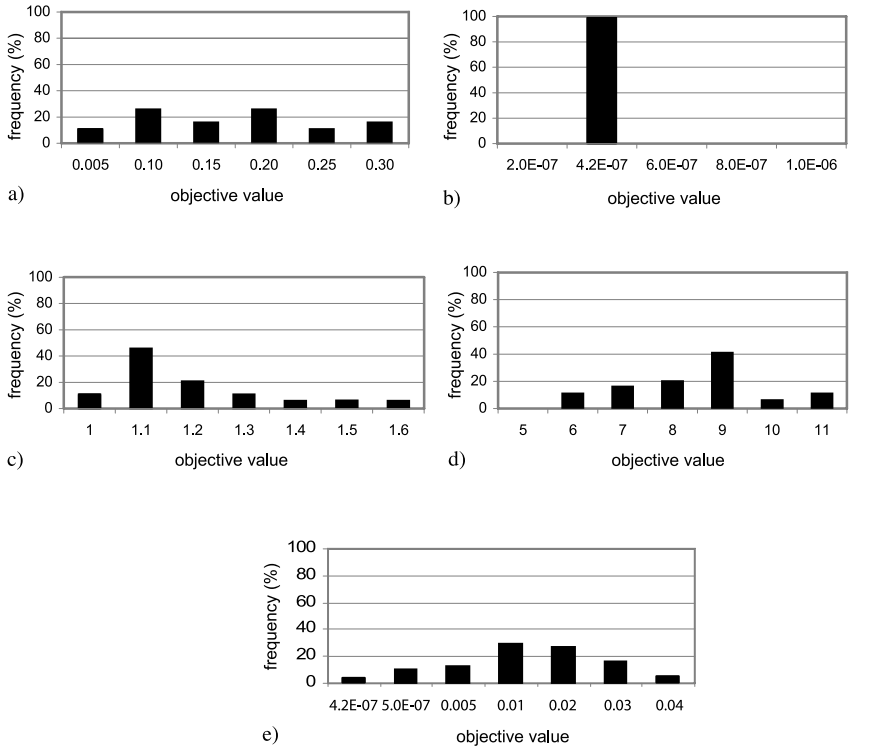

Fig. 9 Histograms drawn from the results obtained for the function F6 by (**a**) TS, (**b**) PTS, (**c**) GA, (**d**) TACO, and (**e**) PACO

A special type of RNN is the Elman network [41]. Elman network and its modified models have been used in applications of system identification. Figure 11 depicts the original Elman network with three layers of neurons. The first layer of this network consists of two different groups of neurons. These are the group of external input neurons and the group of internal input neurons also called context units. Context units are also known as memory units as they store the previous output of the hidden neurons. Elman networks introduced feedback from the hidden layer to the context portion of the input layer. Thus, the Elman network has feedforward and feedback connections. However, so that it can be trained essentially as feedforward networks by means of the simple BP algorithm, the feedback connection weights have to be kept constant. For the training to converge, it is important to select the correct values for the feedback connection weights. However, finding these values manually can be a lengthy trial-and-error process.

In this part of the work, the performance of the proposed PACO algorithm is tested for training the Elman network to identify dynamic plants. The use of the PACO algorithm to train the Elman network to identify a dynamic plant is illustrated in Fig. 12. Here, $y_m(k)$ and $y_p(k)$ are the outputs of the network and plant, at time k,

respectively. Training of the network can be considered as a minimization problem defined by

$$\min_{\mathbf{w}\in W} J(\mathbf{w}) \tag{9}$$

where $\mathbf{w} = [w_1 w_2 w_3 ... w_v]^T$ is the weight vector of the network. The time-averaged cost function $J(\mathbf{w})$ to be minimized by adaptively adjusting $\mathbf{w}$ can be expressed as

$$\min J(\mathbf{w}) = \left(\frac{1}{N} \sum_{k=1}^{N} \left(y_p(k) - y_m(k) \right)^2 \right)^{1/2} \tag{10}$$

where N is the number of samples used for calculation of the cost function.

A solution to the problem is a string of trainable connection weights representing a possible network (Fig. 13). The PACO algorithm searches for the best weight set by means of cost function values calculated for solutions in string form.

The structure of the network employed in this work is selected as in [4] to compare the results. Since the plants to be identified are single-input single-output (SISO), the number of external input neurons and output neurons is equal to one. The number of neurons at the hidden layer is equal to 6. Therefore, the total number of connections is 54, of which 6 are feedback connections. In the case of only

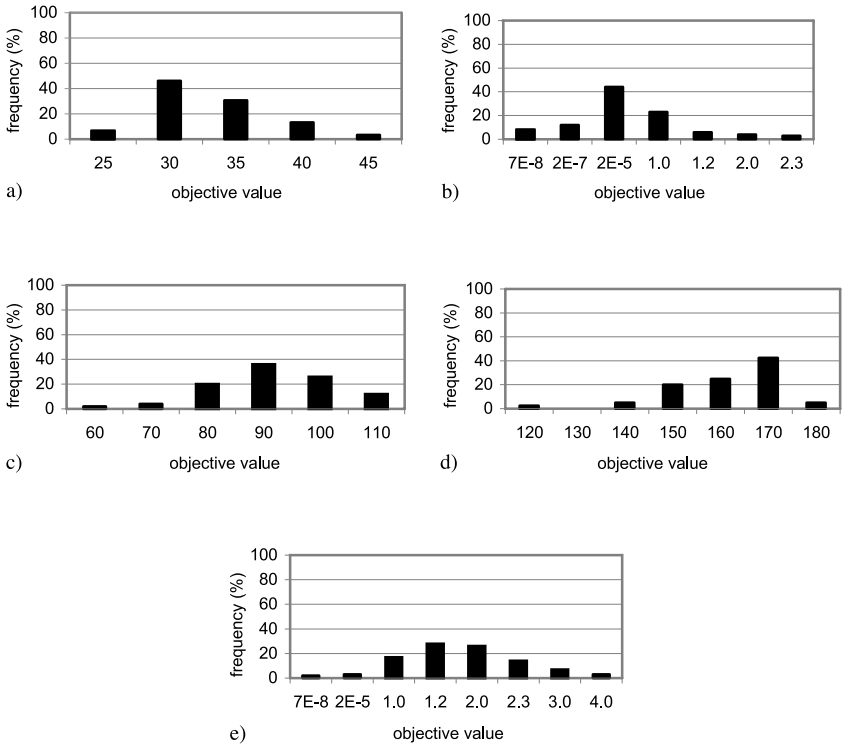

Fig. 10 Histograms drawn from the results obtained for the function F7 by (**a**) TS, (**b**) PTS, (**c**) GA, (**d**) TACO, and (**e**) PACO

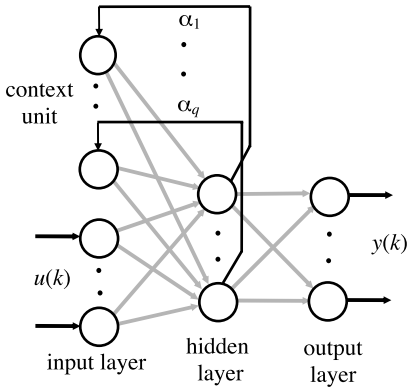

Fig. 11 Structure of the Elman network

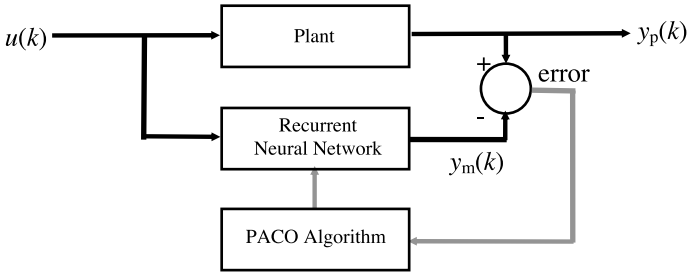

Fig. 12 Scheme for training a network to identify a plant using the PACO algorithm

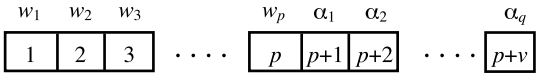

Fig. 13 Representation of the trainable weights of a network in string form

feedforward connections being trainable, a solution is represented as a string of 48 weights. When all connections have trainable weights, then the string consists of 54 weights, of which 6 are feedback connection weights. In both cases, each weight is represented with 16 binary bits. The feedback connections have weight values ranging from 0.0 to 1.0 while feedforward can have positive or negative weights between 1.0 and −1.0. Note that from the point of view of the PACO algorithm, there is no difference between feedback and feedforward connections, and training one type of connections is carried out identically to training the other, unlike in the case of the commonly used BP training algorithm.

In the training stage, first a sequence of input signals $u(k)$, $(k = 0, 1, \dots)$ is fed to both the plant and the recurrent network designed with weights obtained from a solution of the PACO algorithm. Second the rms error value between the plant and recurrent network outputs is computed by means of Equation (10). Next, the rms er-

ror values computed for the solutions are used to select the highest evaluation weight set. The weight set with which the minimum rms error was obtained is selected as the highest evaluation weight set. From the point of view of the optimization, this is again a minimization problem. Simulations were conducted to study the ability of RNN trained by PACO to model a linear and non-linear plant. A sampling period of 0.01 s was assumed in all cases.

Linear plant: This is a third-order linear system described with the following discrete-time equation,

$$y(k) = A_1 y(k-1) + A_2 y(k-2) + A_3 y(k-3) + B_1 u(k-1) + B_2 u(k-2) + B_3 u(k-3) \quad (11)$$

where $A_1 = 2.627771$, $A_2 = -2.333261$, $A_3 = 0.697676$, $B_1 = 0.017203$, $B_2 = -0.030862$, $B_3 = 0.014086$.

The Elman network with all linear neurons was tested. Training input signal, $u(k)$, $k = 0, 1, \ldots, 199$, was randomly produced and varied between -2.0 and 2.0. First the results were obtained by assuming that only the feedforward connection weights are trainable. Second the results were obtained by considering all connection weights of the Elman network trainable. For each case, experiments were repeated six times for different initial solutions. The results obtained using the BP and the PACO algorithms are given in Fig. 14. As an example, the responses of the plant and the network designed by the PACO are presented in Fig. 15. The average rms error values and the improvement percentages for a linear plant obtained using BP and PACO algorithms are presented in Table 3.

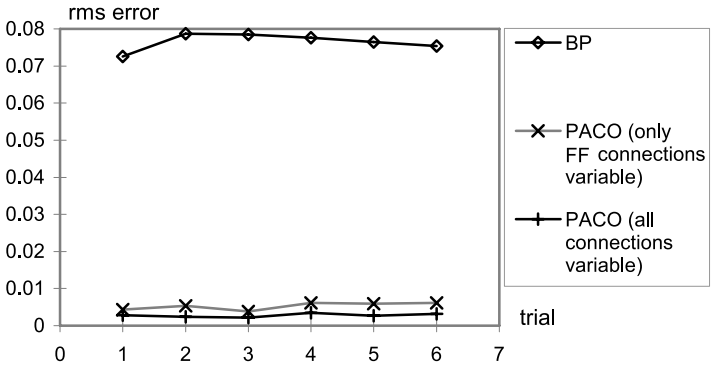

Fig. 14 RMS error values obtained for the linear plant for six runs with different initial solutions

Table 3 Comparison of results for the linear plant

Model	Average rms error	Improvement(%)
Back Propagation (BP)	7.67536×10^{-02}	-
PACO ($\alpha = 1$)	5.26118×10^{-03}	93.15
PACO (all weights trainable)	2.76346×10^{-03}	96.40

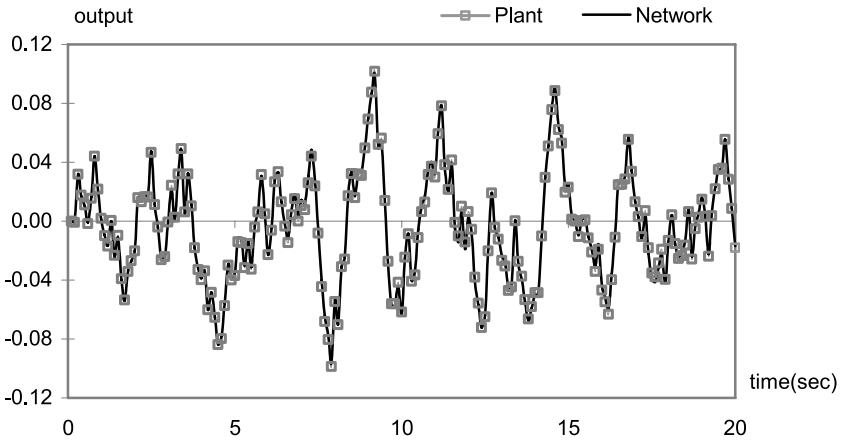

Fig. 15 Responses of the plant and the network trained by the PACO algorithm (third order linear plant, rms error = 2.137717×10^{-03})

Non-linear plant: The second plant model adopted for the simulations was that of a simple pendulum swinging through small angles [42]. The discrete-time description of the plant is:

$$y(k) = \left(2 - \frac{\lambda T}{ML^2}\right) y(k-1) + \left(\frac{\lambda T}{ML^2} - 1 - \frac{gT^2}{L}\right) y(k-2) + \frac{gT^2}{6L} y^3(k-2) - \frac{T^2}{ML^2} u(k-2) \tag{12}$$

where M stands for the mass of the pendulum, L the length, g the acceleration due to gravity, λ the friction coefficient, y the angle of deviation from the vertical position, and u the external force exerted on the pendulum. The parameters used in this model were as follows:

$T = 0.2$ s, $g = 9.8$ m/s^2, $\lambda = 1.2$ kgm^2/s, $M = 1.0$ kg, $L = 0.5$ m.

Replacing the parameters with their values in Equation (12) gives:

$$y(k) = A_1 y(k-1) + A_2 y(k-2) + A_3 y^3(k-2) + B_1 u(k-2) \tag{13}$$

where $A_1 = 1.04$, $A_2 = -0.824$, $A_3 = 0.130667$, $B_1 = -0.16$.

The Elman network with non-linear neurons in the hidden layer was employed. The hyperbolic tangent function was adopted as the activation function of non-linear neurons. The neural networks were trained using the same sequence of random input signals as mentioned above. As in the case of the linear plant, the results were obtained for six different runs with different initial solutions. The rms error values obtained by the BP algorithm and the PACO are presented in Fig. 16. As an example, the responses of the non-linear plant and the recurrent network with the

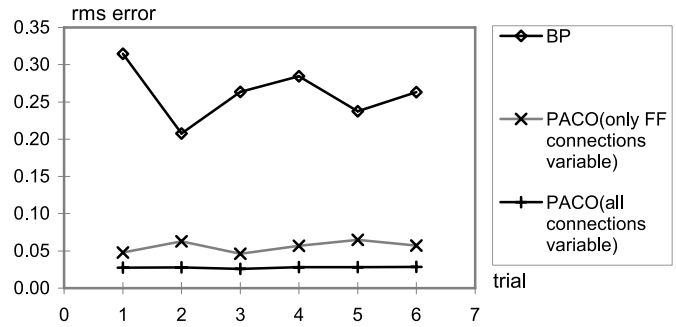

Fig. 16 RMS error values obtained for the non-linear plant for six different runs with different initial solutions

weights obtained by the PACO are shown in Fig. 17. The average rms error values and the improvement percentages for the non-linear plant obtained using BP and PACO algorithms are presented in Table 4.

5 Discussion

From the histograms obtained for the numerical test functions 1 and 3, it is seen that the basic TS, PTS and PACO algorithms are able to find the optimum solution

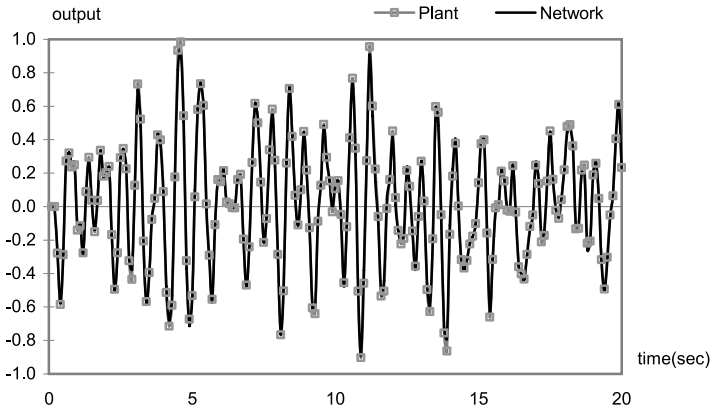

Fig. 17 Responses of the plant and the network trained by the PACO algorithm (non-linear plant, rms error = 2.611416×10^{-02})

Table 4 Comparison of results for the non-linear plant

Model	Average rms error	Improvement(%)
Back Propagation (BP)	0.26182	-
PACO ($\alpha = 1$)	5.59883×10^{-2}	78.62
PACO (all weights trainable)	2.78438×10^{-2}	89.37

for all runs (see Figs. 4 and 6). The reason is that the first problem is a convex and continuous function, and the third is a convex and discrete function. For the rest of the numerical test problems, the basic TS, GA and TACO cannot reach the optimal solution for all runs. However, PTS and the proposed PACO can find the optimum solutions or solutions very near to the optimum for each run. From the histograms presented in Figs. 5, 7 and 8 it can easily be concluded that the proposed PACO is able to find better solutions for F2, F4 and F5 than the basic TS, PTS, GA and TACO. Although the PACO can provide better solutions for the F6 and F7 than the basic TS, GA and TACO algorithms, its performance is not as good as the PTS algorithm. However, the results obtained by the PACO for these functions are also acceptable.

The original Elman network could identify the third-order linear plant successfully. Note that an original Elman network with an identical structure to that adopted for the original Elman network employed in this work and trained using the standard BP algorithm failed to identify even a second-order linear plant. Moreover, when the original Elman network had been trained by the basic GA, the third-order plant could not be identified although the second-order plant had been identified successfully [43]. It can be seen that, apparently for both plants, the training was significantly more successful when all connection weights of the network were trainable than when only feedforward connection weights could be changed. Thus, by using the PACO algorithm not only was it possible and simple to train the feedback connection weights, but the training time required was lower than for the feedforward connection weights alone. It is clearly seen from Figs. 14 and 16 and Tables 3 and 4 that, for both network structures (with all connection weights variable and with only feedforward connection weights trainable), the proposed PACO trained the networks better than the BP algorithm.

In this work, the data representation structure of the TACO and the search strategy of ACS were employed in the proposed model. However, the proposed model is a general structure for parallelization and it is also possible to use other ant based search strategies such as MAX-MIN ant system. Moreover, performance of the proposed model was tested on continuous problems; however this model can be implemented for combinatorial type problems. Using different ant based search strategies and performance examination of combinatorial problems are considered as future studies for the proposed model.

6 Conclusions

In this study a parallel ant colony optimization algorithm was proposed. The performance of the proposed algorithm was compared with that of basic TS, PTS, GA and TACO algorithms for numerical test problems. It was also applied to training a recurrent neural network to identify linear and non-linear plants, and the results obtained were compared with those produced by the BP algorithm. From the simulation results it was concluded that the proposed algorithm can be used to search multi-modal spaces successfully, and can be efficiently applied to train recurrent neural networks to identify dynamic plants accurately. It can be finally concluded that the PACO algorithm might be an efficient tool for solving continuous optimization problems.

References

1. Reeves CR (Ed.) (1995) Modern Heuristic Techniques for Combinatorial Optimization. McGraw-Hill: UK.
2. Corne D, Dorigo M, Glover F (Eds) (1999) New Ideas in Optimization, McGraw-Hill: UK.
3. Farmer JD, Packard NH, Perelson AS (1986) The Immune System, Adaptation, and Machine Learning. Physica, 22D:187–204
4. Kalinli A, Karaboga D (2004) Training recurrent neural networks by using parallel tabu search algorithm based on crossover operation. Engineering Applications of Artificial Inteligence, 17(5):529–542
5. Dorigo M, Maniezzo V, Colorni A (1991) Positive feedback as a search strategy. Technical Report No:91–016 Politecnico di Milano
6. Dorigo M, Maniezzo V, Colorni A (1996) The ant system: Optimization by a colony of cooperating agents. IEEE Trans. on Systems, Man and Cybernetics – Part B, 26(1):1–13
7. Christopher FH et al. (2001) Swarm intelligence: an application of social insect optimization techniques to the traveling salesman problem. Artificial Intelligence I
8. Bullnheimer B, Hartl RF, and Strauss C (1999) A new rank based version of the ant system, a computational study. Central European J for Operations Research and Economics, 7(1):25–38
9. Stützle T, Hoos HH (1997) The MAX-MIN ant system and local search for the traveling salesman problem. In Baeck T, Michalewicz Z, Yao X, (Eds), Proc. of the IEEE Int. Conf. on Evolutionary Computation (ICEC'97):309–314
10. Gambardella LM, Dorigo M (1996) Solving symmetric and asymmetric TSPs by ant colonies. Proc. of IEEE Int. Conf. on Evolutionary Computation, IEEE-EC 96, Nagoya, Japan:622–627
11. Di Caro G, Dorigo M (1998) Mobile agents for adaptive routing. Proc. of 31st Hawaii Conf. on Systems Sciences (HICSS-31):74–83
12. Stützle T, Dorigo M (1999) ACO algorithms for quadratic assignment problem. in: Corne D, Dorigo M, Glover F (Eds), New Ideas in Optimization, McGraw-Hill:33–50
13. Gambardella LM, Taillard E, Agazzi G (1999) MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. Technical Report, IDSIA-06: Switzerland
14. Bilchev G, Parmee IC (1995) The ant colony metaphor for searching continuous design spaces. Lecture Notes in Computer Science, Springer-Verlag, LNCS 993:25–39
15. Monmarché N, Venturini G, Slimane M (2000) On how Pachycondyla apicalis ants suggest a new search algorithm. Future Generation Systems Computer 16(8):937–946
16. Dreo J, Siarry P (2004) Continuous ant colony algorithm based on dense heterarchy. Future Generation Computer Systems, 20(5):841–856
17. Hiroyasu T, Miki M, Ono Y, Minami Y (2000) Ant colony for continuous functions, The Science and Engineering, Doshisha University
18. Bullnheimer B, Kotsis G, Strauss C (1998) Parallelization strategies for the ant system. in: De Leone R, Murli A, Pardalos P, Toraldo G (Eds), High Performance Algorithms and Software in Nonlinear Optimization. Kluwer Series of Applied Optimization, Kluwer Academic Publishers, Dordrecht, The Netherlands, 24:87–100
19. Stützle T (1998) Parallelization strategies for ant colony optimization, in: Eiben AE, Back T, Schoenauer M, Schwefel HP (Eds), Fifth Int. Conf. on Parallel Problem Solving from Nature, Springer-Verlag: 1498:722–731

20. Middendorf M, Reischle F, Schmeck H (2000) Information exchange in multicolony algorithms. in: Rolim J, Chiola G, Conte G, Mansini LV, Ibarra OH., Nakano H. (Eds), Parallel and Distributed Processing: 15 IPDPSP Workshops Mexico, Lecture Notes in Computer Science, Springer-Verlag, Heidelberg, Germany, 1800:645–652
21. Dorigo M (1993) Parallel ant system: An experimental study. Unpublished manuscript, (Downloadable from http://iridia.ulb.ac.be/~mdorigo/ACO/ACO.html)
22. Talbi EG, Roux O, Fonlupt C, Robillard D (1999) Parallel ant colonies for combinatorial optimization problems. in: Rolim J. et al. (Eds) Parallel and Distributed Processing, 11 IPPS/SPDP'99 Workshops, Lecture Notes in Computer Science, Springer-Verlag, London, UK 1586:239–247
23. Bolondi M, Bondanza M (1993) Parallelizzazione di un algoritmo per la risoluzione del problema del commesso viaggiatore. Master's Thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano: Italy
24. Michel R, Middendorf M (1998) An island model based ant system with lookahead for the shortest supersquence problem. in: Eiben AE, Back T, Schoenauer H, Schwefel P (Eds), Parallel Problem Solving from the Nature, Lecture Notes in Computer Science, Springer-Verlag, Heidelberg, Germany, 1498:692–701
25. Delisle P, Krajecki M, Gravel M, Gagné C (2001) Parallel implementation of an ant colony optimization metaheuristic with openmp. Int. Conf. on Parallel Architectures and Compilation Techniques, Proceedings of the 3rd European Workshop on OpenMP (EWOMP'01), Barcelona, Spain
26. Krüger F, Merkle D, Middendorf M (1998) Studies on a parallel ant system for the BSP model, unpublished manuscript. (Downloadable from http://citeseer.ist.psu.edu/239263.html)
27. De Jong KA (1975) An Analysis of The Behaviour of a Class of Genetic Adaptive Systems. PhD thesis, University of Michigan
28. Pham DT, Liu X (1999) Neural Networks for Identification. Prediction and Control, 4th edn, Springer-Verlag
29. Arifovic J, Gencay R (2001) Using genetic algorithms to select architecture of a feedforward artificial neural network. Physica A, 289:574–594
30. Sexton RS, Gupta JND (2000) Comparative evaluation of genetic algorithm and backpropagation for training neural networks. Information Sciences, 129:45–59
31. Castillo PA, Merelo JJ, Prieto A, Rivas V, Romero G (2000) G-Prop: Global optimization of multilayer percetptrons using Gas. Neurocomputing, 35:149–163
32. Ku KW, Mak MW, Siu WC (1999) Adding learning to cellular genetic algorithms for training recurrent neural networks. IEEE Trans. on Neural Networks, 10(2):239-252
33. Blanco A, Delgado M, Pegalajar MC (2000) A genetic algorithm to obtain the optimal recurrent neural network. Int. J. Approximate Reasoning, 23:67–83
34. Blanco A, Delgado M, Pegalajar MC (2001) A real-coded genetic algorithm for training recurrent neural networks. Neural Networks, 14:93–105
35. Castillo PA, Gonzalez J, Merelo JJ, Prieto A, Rivas V, Romero G (1999) SA-Prop: Optimization of multilayer perceptron parameters using simulated annealing. Lecture Notes in Computer Science, Springer, 606:661-670
36. Sexton RS, Alidaee B, Dorsey RE, Johnson JD (1998) Global optimization for artificial neural networks: A tabu search application. European J of Operational Research, 106:570–584
37. Battiti R, Tecchiolli G (1995) Training neural nets with the reactive tabu search. IEEE Trans. on Neural Networks, 6(5):1185–1200
38. Zhang S-B, Liu Z-M (2001) Neural network training using ant algorithm in ATM traffic control. IEEE Int. Symp. on Circuits and Systems (ISCAS 2001) 2:157–160

39. Blum C, Socha K (2005) Training feed-forward neural networks with ant colony optimization: An application to pattern classification. Fifth Int. Conf. on Hybrid Intelligent Systems
40. Li J-B, Chung Y-K (2005) A novel back-propagation neural network training algorithm designed by an ant colony optimization. Transmission and Distribution Conference and Exhibition: Asia and Pacific:1–5
41. Elman JL (1990) Finding structure in time. Cognitive Science, 14:179–211
42. Liu X (1993) Modelling and Prediction Using Neural Networks. PhD Thesis, University of Wales College of Cardiff, Cardiff, UK.
43. Pham DT, Karaboga D (1999) Training Elman and Jordan networks for system identification using genetic algorithms. J. of Artificial Intelligence in Engineering 13:107–117