New Chaotic PSO-Based Neural Network Predictive Control for Nonlinear Process

Ying Song, Zengqiang Chen, and Zhuzhi Yuan

***Abstract*—In this letter, a novel nonlinear neural network (NN) predictive control strategy based on the new tent-map chaotic particle swarm optimization (TCPSO) is presented. The TCPSO incorporating tent-map chaos, which can avoid trapping to local minima and improve the searching performance of standard particle swarm optimization (PSO), is applied to perform the nonlinear optimization to enhance the convergence and accuracy. Numerical simulations of two benchmark functions are used to test the performance of TCPSO. Furthermore, simulation on a nonlinear plant is given to illustrate the effectiveness of the proposed control scheme.**

***Index Terms*—Neural network (NN), nonlinear plant, particle swarm optimization (PSO), predictive control, tent-map chaos.**

I. Introduction

Generalized predictive control (GPC) [1] technique has been implemented successfully and widely in industrial processes [2]. The design of a GPC system relies heavily upon an explicit mathematical model of the system under control. In many cases, such a model is very difficult to build due to, for instance, lack of knowledge about the system or the presence of strong nonlinear dynamics in the behavior of the system. The conventional controllers are designed based on the linearized model. When the nonlinearity is strong, such controllers may produce big errors or even be out of control.

Neural networks (NN) have been widely studied and applied to various research fields such as nonlinear modeling and control [3], pattern recognition [4], chaos generation [5], etc. For nonlinear plants, the ability of the GPC to make accurate predictions can be enhanced if an NN is used to learn the plant dynamics instead of standard nonlinear modeling techniques. The NN-based predictive control (NNPC) strategies have been found to be effective in controlling a wide class of nonlinear processes in the past [6]–[11]. Different ways of neural model being embedded in predictive control systems were reviewed by Hussian [6]. In the NNPC, the NN is used as the prediction model of the nonlinear plant and the system performance is greatly dependent upon the online optimization. Several algorithms were successfully implemented in the NNPC, such as gradient–descent [7]–[9] and Newton–Raphson methods [10]. The Jacobian and Hessian matrix used for solving the optimization are normally formulated in terms of the structure of the NN. However, these gradient-based optimization methods usually provide local optima, require the NNPC cost function differential, and they are still a complex procedure for calculating the Jacobian and Hessian matrix even under some simplifications; hence, the intelligent evolutionary algorithms are more suitable for the optimizing in NNPC, such as the genetic algorithm (GA) [11] and particle swarm optimization (PSO) [20], [21]. In addition, specifying the weights of an NN is mostly viewed as an optimization process where the goal of the computation is to find an optimal value of an error function.

Manuscript received December 13, 2005; revised April 8, 2006 and October 3, 2006; accepted November 5, 2006. This work was supported in part by the Natural Science Foundation of China under Grants 60374037 and 60574036, by the Program for New Century Excellent Talents in University of China (NCET), by the Specialized Research Fund for the Doctoral Program of Higher Education of China under Grant 20050055013, and by the Opening Project Foundation of National Laboratory of Industrial Control Technology under 0708008.

The authors are with the Department of Automation, Nankai Unversity, Tianjin 300071, China (e-mail: songyingnk@yahoo.com.cn; chenzq@nankai.edu.cn; yuanzhzh@nankai.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNN.2006.890809

PSO [12] is newly developed evolutionary technique. Due to the simple concept, easy implementation, and quick convergence, nowadays, PSO has gained much attention and wide applications in different fields. However, the standard PSO greatly depends on its parameters and exists as the premature phenomenon, especially in solving complex multihump problems [13]. Chaos is a kind of characteristic of nonlinear systems. A chaotic motion can traverse every state in a certain region by its own regularity, and every state is visited only once. Due to the unique ergodicity and special ability to avoid being trapped in local optima, chaos search is much higher than some other stochastic algorithms [16]. However, it often needs a large number of iterations to reach the global optimum and is not effective in large searching space. Recently, several attempts for PSO using chaos methods were made [17]–[20]. Xie [17] introduced chaos into the system by randomly reinitializing the particle positions with a small constant probability, while Iqbal [18] improved it with a time-dependence to the probability of reinitialization. Liu [19] incorporated chaos into PSO with adaptive inertia weight factor to construct a chaotic PSO. These researches are almost based on logistic map to generate the chaos variables.

In this letter, a new tent-map-based chaotic particle swarm optimization (TCPSO) is proposed, and it is utilized as the optimization technique to obtain optimal future control inputs for neural network predictive control (NNPC). Moreover, the TCPSO is also adopted for training the NN which is used as the prediction model in NNPC. The tent map is studied in the mathematics of dynamical systems because it has several interesting properties such as chaotic orbits, simple shape, and so on. Most importantly, tent map shows the outstanding advantages and higher iterative speed than the logistic map, because the invariant density function for tent map is the uniform function while the invariant density function for logistic map is Chebyshev-type function.

II. Standard PSO

PSO [12] is an evolutionary algorithm to simulate the movement of flocks of birds. Like other population-based search algorithms, PSO is initialized with a swarm of random solutions (particles). Each particle *flies* in D-dimensional problem space with a velocity, which is adjusted at each time step. The particle flies towards a position, which depends on its own past best position and the position of the best of its neighbors. The quality of a particle position depends on a problem-specific objective function (fitness).

The position of the ith particle is presented by a vector $X_i = (x_{i1}, \ldots, x_{id}, \ldots, x_{iD})$, where $x_{id} \in [x_{\min,d}, x_{\max,d}]$, $d = 1, \ldots, D$. $x_{\min,d}$ and $x_{\max,d}$ are the lower and upper bounds for the dth dimension, respectively. The best position (i.e., that with the best fitness, the so-called *pbest*) of particle i is recorded as $P_i = (p_{i1}, \ldots, p_{id}, \ldots, p_{iD})$. Similarly, the location of the best particle among the population is recorded by the index g and the location P_g is called *gbest* (global best). The velocity of the ith particle $V_i = (v_{i1}, \ldots, v_{id}, \ldots, v_{iD})$, is limited to a maximum velocity $V_{\max} = (v_{\max,1}, \ldots, v_{\max,d}, \ldots, v_{\max,D})$. At each time step, the particles' positions are updated depending on their *pbest* and *gbest* according to

$$v_{id}^{k+1} = \omega v_{id}^k + c_1 r_1 \left(p_{id}^k - x_{id}^k\right) + c_2 r_2 \left(p_{gd}^k - x_{id}^k\right) \quad (1)$$

$$x_{id}^{k+1} = x_{id}^k + v_{id}^{k+1} \quad (2)$$

where k is the iteration index and ω is the inertia weight, determining how much of the previous velocity of the particle is preserved. c_1 and c_2 are two positive acceleration coefficients, generally $c_1 = c_2 = 2$. r_1 and r_2 are two uniform random numbers samples from $U(0,1)$. For the velocity update equation, the second part represents the private thinking by itself; the third part is the social part, which represents the cooperation among the individuals. A standard PSO algorithm is detailed in [12]–[15].

III. TCPSO

Studies by Angeline [13] showed that although the standard PSO discovered reasonable quality solutions much faster than other evolutionary algorithms, it did not possess the ability to improve upon the quality of the solutions as the number of generations was increased. That is, it is lacking enough capability to achieve *sustainable development* [17]. The swarm becomes stagnated after a certain number of iterations.

A. Chaotic PSO

To enrich the search behavior, chaotic dynamics is incorporated into the PSO (CPSO) [16], [20]. The logistic map is usually employed for constructing CPSO. The logistic map is defined by

$$z_{n+1} = 4z_n(1 - z_n), \qquad 0 \le z_0 \le 1 \tag{3}$$

where $n = 0, 1, 2, \ldots$. A minute difference in the initial value of the chaotic variable would result in a considerable difference in its long-time behavior. The track of chaotic variable can travel ergodically over the whole search space. A general procedure of chaotic search can be found in [16].

Based on the standard PSO and the chaotic search, the procedure of CPSO is described in [19].

B. Tent-Map-Based Chaotic PSO

Chaos variables are almost generated by the logistic map in the literature. However, the invariant density of the iterates for logistic map is

$$\rho(x) = \frac{1}{2\pi\sqrt{x(1-x)}}. \tag{4}$$

It is Chebyshev-type distribution function in the interval [0, 1], which affects the global search capacity and computational efficiency of chaos search severely. The invariant density of the iterates for tent map is $\rho(x) = 1$. Since it is the uniform distribution function in the interval [0, 1], tent map shows the outstanding advantages and higher iterative speed than logistic map. In this letter, we use tent map to generate chaos variables. The tent map is defined by

$$z_{n+1} = \mu(1 - 2|z_n - 0.5|), \qquad 0 \le z_0 \le 1 \tag{5}$$

where $\mu \in [0,1]$ is the bifurcation parameter. Specially, when $\mu = 1$, the tent map exhibits entirely chaotic dynamics and ergodicity in the interval [0, 1].

According to the chaotic search, a chaotic disturbance can be added as follows:

$$Z_k^{'} = (1-\beta)\Psi^* + \beta Z_k$$

where Ψ^* is the optimal chaos vector where the current optima $X^* = (x_1^*, \ldots, x_D^*)$ are mapped into the interval $[0,1]$, $\Psi^* = (X^* - X_{\min})/(X_{\max} - X_{\min})$. $Z = (z_1, \ldots, x_D)$ is the chaos vector which is generated by tent map; $Z^{'} = (z_1^{'}, \ldots, x_D^{'})$ is the chaos vector corresponding to $X = (x_1, \ldots, x_D)$ where the disturbance has been added. β is a parameter, $0 \le \beta \le 1$. Here, $\beta(k) = 1 - k/k_{\max}$, where $k_{\max}$ is the maximal iteration. The adaptive mechanism can meet the different requirement for β during the different search stages. That is, a large β is expected to change X largely at the initial stage, while a small β is expected to search within the small interval of X^* as X close to the best solution X^*.

The TCPSO algorithm is described as follows.

Step 1) Initialize the swarm ($k = 0$). Use the tent map ($\mu = 1$) to generate the chaos variables and rewrite (5) as

$$z_j^{(i+1)} = 1 - 2|z_j^{(i)} - 0.5|, \qquad j = 1, 2, \ldots, D \tag{7}$$

where z_j denotes the jth chaos variable, and i denotes the chaos iteration number. According to [16], set $i = 0$ and generate D chaos variables by (7). Then, let $i = 1, 2, \ldots, m$ in turn and generate the initial swarm.

Step 2) Map the above chaos variable $z_j^{(i)}, i = 1, \ldots, m$ into the search range of decision variable $x_j, j = 1, \ldots, D$

$$x_{ij} = x_{\min,j} + z_j^{(i)}(x_{\max,j} - x_{\min,j}). \tag{8}$$

Define $X_i = (x_{i1}, x_{i2}, \ldots, x_{iD}), i = 1, 2, \ldots, m$.

Step 3) Evaluate the fitness f_i and update *pbest* P_i and *gbest* P_g if needed.

Step 4) Change the velocity V_i and position X_i of the particle according to (1) and (2), respectively.

Step 5) Rank in an ascending order the swarm by fitness. If the current optimum $f(P_g)$ keeps the same in some iterations and the current iteration $k \le (2/3)k_{\max}$, go to step 6); otherwise, turn to step 8).

Step 6) Introduce the chaotic disturbance and chaotic search to update the best particle. Through (6), a chaotic disturbance is added to the bottom 50% of particles which are relatively inferior in the swarm; then, they are mapped into decision variables by (8) and calculated by chaos iterations until a stopping criterion of chaos search is satisfied.

Step 7) Rerank the swarm by fitness and obtain the average fitness $f_{\text{avg}}(X)$ compared with $f(P_g)$. If (9) or stopping criterion is satisfied, stop; otherwise, go to step 8)

$$|f_{\text{avg}}(X) - f(P_g)| \le \varepsilon \tag{9}$$

where $f_{\text{avg}}(X) = 1/m \sum_{i=1}^{m} f(X_i), \varepsilon$ is a small positive constant.

Step 8) Let $k := k + 1$ and go back to step 3).

C. Benchmark Function Tests

In order to test the performance of TCPSO algorithm, minimize two multimodal functions [12]: one is the Rosenbrock function (f_1) and the other is the generalized Griewank function (f_2)

$$f_1(x) = \sum_{i=1}^{D-1} 100\left(x_{i+1} - x_i^2\right) + (x_i - 1)^2, \qquad x_i \in [-30, 30]$$

$$f_2(x) = \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \sum_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1, \qquad x_i \in [-600, 600]$$

For both functions, $v_{\max} = x_{\max}, c_1 = c_2 = 2, m = 20, k_{\max} = 2000$. The fitness value is set as the function value.

First, we compare the TCPSO with the standard PSO, PSO with mutation (MPSO)[15], CPSO [19], and GA. For each function, different dimensions (10, 20, and 30) are tested. ω is fixed as 0.5 in all PSO algorithms. The GA is real code, crossover probability $p_c = 0.8$, and mutation probability $p_m = 0.1$. Table I lists the mean minimum of 50 independent runs. Figs. 1 and 2 show the performances for f_1 and f_2 functions with 30 dimensions.

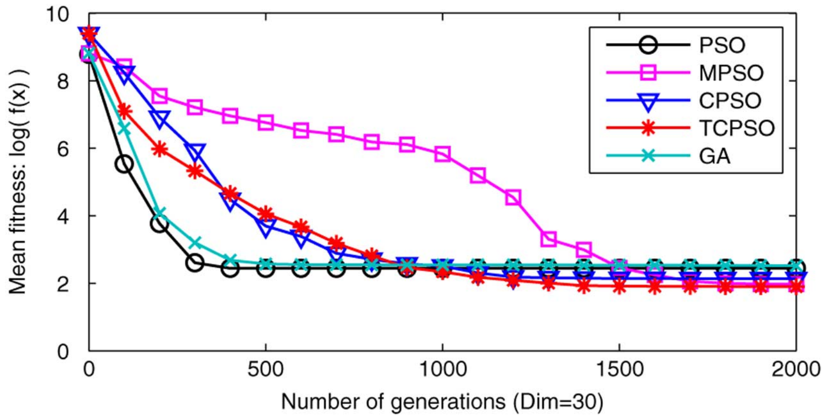

Fig. 1. Performance of the algorithms for 30-dimensional f_1 function.

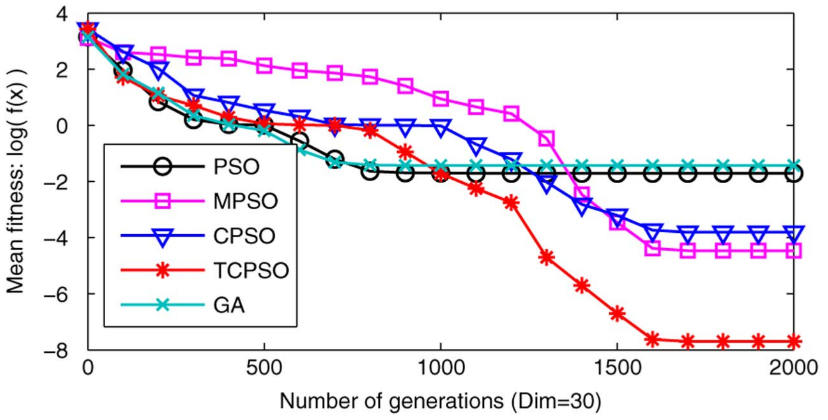

Fig. 2. Performance of the algorithms for 30-dimemsional f_2 function.

TABLE I
MEAN FITNESS FOR f_1 AND f_2 FUNCTIONS

f	D	PSO	MPSO	CPSO	TCPSO	GA
f_1	10	26.2201	10.6114	7.7192	4.1497	36.3015
	20	68.9187	16.0597	17.3572	14.8649	82.8531
	30	277.7157	95.7392	136.3357	79.8258	333.8259
f_2	10	0.0787	0.0517	0.0639	0.0221	0.0935
	20	0.0320	0.0027	0.0074	3.4806e-7	0.0663
	30	0.0197	3.4099e-5	4.6219e-4	1.6704e-8	0.0369

From Table I and Figs. 1 and 2, it can be seen that the results of TCPSO have better accuracy, and TCPSO is superior to PSO, MPSO, CPSO, and GA in terms of the searching solutions. In addition, the chaotic PSOs (including CPSO and TCPSO) are similar to the MPSO, but their mutation mechanisms are different.

Further, to investigate the effect of inertia weight ω on searching quality, Figs. 3 and 4 show the mean fitness value of the best particle found with different ω for f_1 and f_2 functions with 30 dimensions. (For 10 and 20 dimensions, there are the same performances.) The TCPSO is more effective and robust than PSO if the inertia weight is set as a small value ($\omega \in [0, 0.6]$). These performances are consistent with Xie [17].

Comparing TCPSO with GA, standard PSO, MPSO, and CPSO, it shows that TCPSO is superior to the standard PSO, MPSO, CPSO, and GA in terms of the searching solutions. Moreover, the TCPSO is more effective and robust than the mentioned algorithms when the inertia weight is fixed at a small value ($\omega \in [0, 0.6]$).

IV. NNPC

In contrast to the NN direct control, the NNPC is more practical. The core problem in NNPC is the online optimization of the cost index. Some related works have been performed [7]–[10], where nonlinear programming is needed to obtain a solution. These gradient-based methods require vast cost in the complex calculation of the Hessian or Jocabian matrices. In order to reduce the computation load, the gradient-free methods are more appropriate and flexible. In this paper, the TCPSO algorithm is applied to perform the nonlinear optimization in NNPC to enhance the convergence and accuracy.

Assume that the unknown nonlinear system is expressed as the input–output form by

$$y(t)=g(y(t-1),\ldots,y(t-n_a),u(t-\tau-1),\ldots,u(t-\tau-n_b)) \quad (10)$$

where $y(t)$ and $u(t)$ are the output and input of the system, respectively, $g(\cdot)$ is the unknown nonlinear function to be estimated by an NN, n_a and n_b are the orders of the system, and τ is the plant delay as an integer number.

The purpose of NNPC is to select signal $u(t)$, such that the output of the system $y(t)$ is made as close as possible to be a prespecified set-point $r(t)$. A schematic of the NNPC is illustrated in Fig. 5. The control input is calculated to minimize a cost function J at each sampling instant t

$$J = \sum_{j=N_1}^{N_2} [y_r(t+j) - \hat{y}(t+j)]^2 + \lambda \sum_{j=1}^{N_u} [\Delta u(t+j-1)]^2 \quad (11)$$

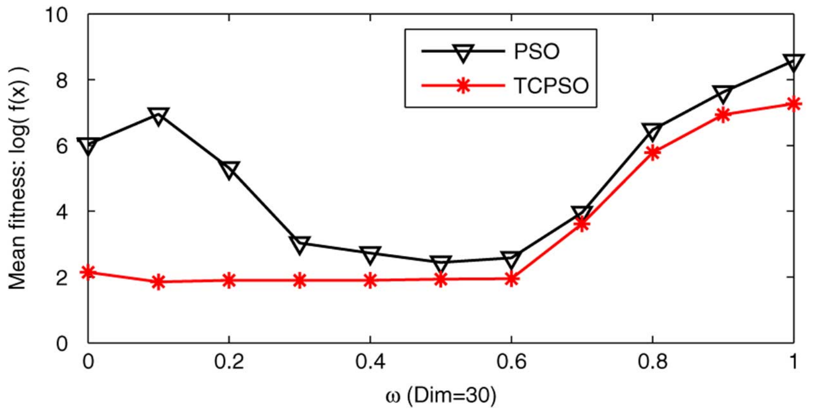

Fig. 3. The 30-dimensional f_1 Rosenbrock function with different ω.

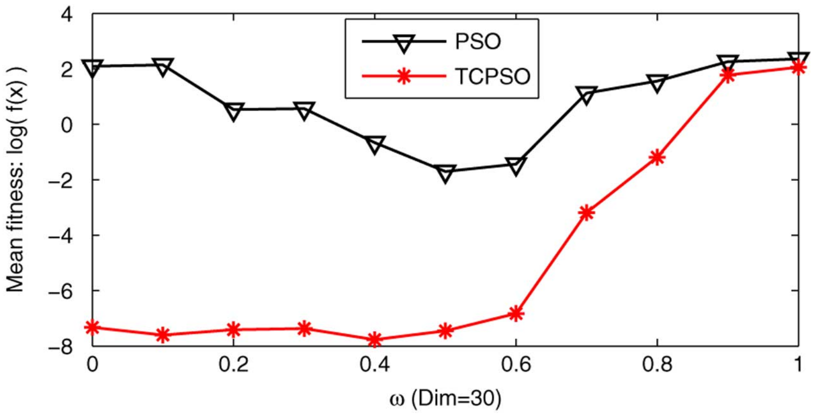

Fig. 4. The 30-dimensional f_2 Griewank function with different ω.

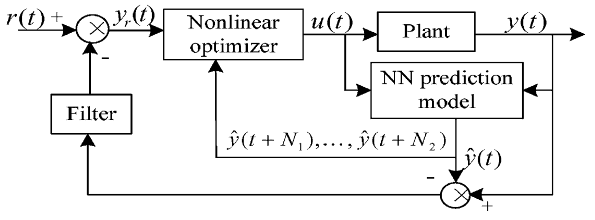

Fig. 5. NN predictive control scheme.

where N_1 and N_2 are the prediction horizon $(N_2 > N_1)$, N_u is the control horizon, generally $N_1 \le N_u \le N_2$, λ is the control weighting factor, $\hat{y}(t+j)$ is the j-step-ahead predicted output by the NN prediction model, $u(t+j)$ is the j-step-ahead future control signal, $y_r(t+j)$ is the j-step-ahead future reference output, which is obtained as following:

$$\begin{cases} y_r(t+j) = \alpha y_r(t+j-1) + (1-\alpha) r(t) \\ y_r(t) = y(t), \qquad j = 1, \ldots, N_2 \end{cases} \tag{12}$$

where α is a soft factor, $0 \le \alpha \le 1$.

A. NN Prediction Model

Since an NN will be used to model the nonlinear plant, the configuration of the network architecture should be considered. A three-layer feedforward NN, which only has one hidden layer, is used to learn the nonlinear plant, since it has been proved that this is sufficient to represent any nonlinear function providing enough nodes are present. The activation functions are hyperbolic tangent for the hidden layer and linear for the output layer.

Since the input to the NN is $\phi = [y(t-1), \ldots, y(t-n_a), u(t-\tau-1), \ldots, u(t-\tau-n_b)]^T$, the neural model for the unknown system (10) can be expressed as

$$\hat{y}(t) = \sum_{j=1}^{N_H} w_j^o \cdot \tanh(\mathrm{net}_j(t)) + b^o \tag{13}$$

$$\mathrm{net}_j(t) = \sum_{i=1}^{n_a} w_{j,i}^I y(t-i) + \sum_{i=1}^{n_b} w_{j,i+n_a}^I u(t-\tau-i) + b_j^I \tag{14}$$

where $\hat{y}(t)$ is the NN output, $\mathrm{net}_j(t)$ is the activation level of the jth node's output function, n_H is the number of hidden nodes in the hidden layer, w_j^o is the weight connecting the jth hidden node to the output node and $w_{j,i}^I$ the weight connecting the ith input node to the jth hidden node, b_j^I is the bias on the jth hidden node, and b^o is the bias on the output node.

Training algorithm is critical to NN, which will affect the NN's prediction and generalization capability. Since backpropagation (BP) algorithm is easily trapped in local minima and its convergent performance greatly depends on its learning rate parameter and the initial conditions, the proposed TCPSO algorithm is employed to train the NN. During the training, the weights and biases of the NN are optimized by the TCPSO which guarantees the mean-square-error (mse) criterion: $\mathrm{mse} = (1)/(2p) \sum_{t=1}^{p} [y(t) - \hat{y}(t \mid W)]^2$, where p is the training patterns and W is the optimized vector containing all the neural weights and biases.

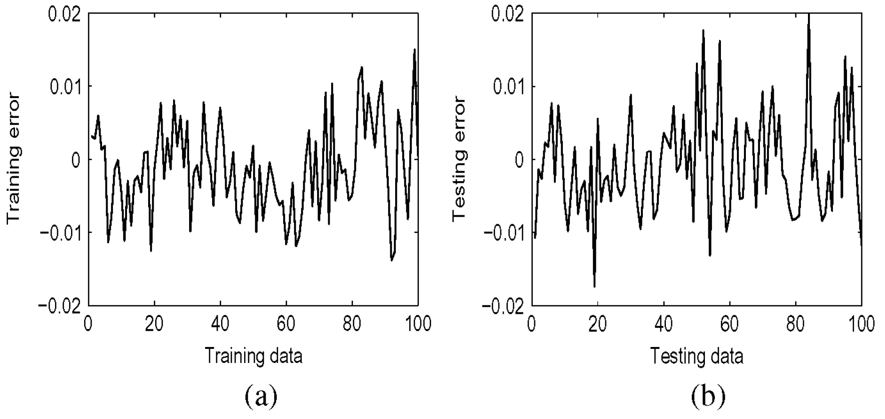

Fig. 6. (a) NN training error. (b) NN test error.

B. Prediction Using the Trained NN

The NNPC uses the output of the NN model to predict the plant's dynamics to an arbitrary input from the current time t to some future time $t+k(k=1,\ldots,N_2)$. Based on (13)and (14), the k-step-ahead prediction outputs are

$$\hat{y}(t+k)=\sum_{j=1}^{N_H} w_j^o \cdot \tanh(\mathrm{net}_j(t+k))+b^o \tag{15}$$

$$\begin{aligned}\mathrm{net}_j(t+k)=b_j+&\sum_{i=1}^{\min\{k,n_a\}} w_{j,i}^I \hat{y}(t+k-i)+\sum_{i=k+1}^{n_a} w_{j,i}^I y(t+k-i)\\ &+\sum_{i=1}^{n_b} w_{j,i+n_a}^I u(t+k-i).\end{aligned} \tag{16}$$

The first summation of (16) handles the recursive part of prediction. This feeds back the NN output $\hat{y}$, for k or n_a times, which is ever smaller. The second summation of (16) handles the previous values of y. The last summation of (16) handles the previous future values of u up to $u(t+k)$. Note that $u(t+k)$ is equal to $u(t+N_u-1)$ when $k<N_u-1$.

C. NNPC Based on the TCPSO Algorithm

The NNPC algorithm has the following important steps.

Step 1) At time step t, simultaneously sample inputs and outputs of process [including $y(t)$].

Step 2) Start with the previous calculated control input vector, and compute the predictive output values using the NN model, hence, determine the cost function J.

Step 3) Use the optimization algorithm to calculate a new control input that minimizes the cost function.

Step 4) Repeat steps 2) and 3) until desired minimization is achieved.

Step 5) Send the first control input to the process.

Step 6) Repeat entire procedure for each time step.

To use the proposed TCPSO algorithm as the optimization algorithm within the predictive control framework, it is necessary to modify it accordingly. The prediction steps are represented by population particles. Thus, control actions $\Delta\mathbf{u}=[\Delta u(t),\Delta u(t+1),\ldots,\Delta u(t+N_u-1)]^T$ to be applied to the system in a specified future time are encoded into corresponding data structures that form the population. The fitness function is the optimization objective function J (11). In each generation, the best two solutions found are shifted one position toward the present instant and introduced in the population of the next generation.

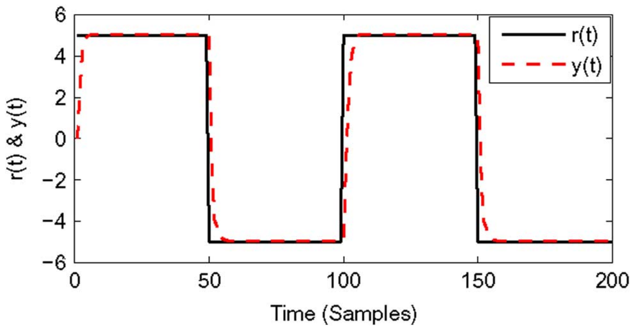

Fig. 7. Performance of NNPC based on TCPSO for square signal.

V. Simulation Studies

To demonstrate the control performance of the TCPSO-based NNPC, consider the following nonlinear plant [9]:

$$y(t)=\frac{y(t-1)y(t-2)[y(t-1)+2.5]}{1+y(t-1)^2+y(t-2)^2}+u(t-1) \tag{17}$$

where $n_a=2, n_b=1$, and $\tau=0$.

The plant is modeled by a feedforward NN with 3–5–1 structure. The input signal applied to plant (17) is a finite sequence of uniformly distributed random variables with range $[-1,1]$. Thus it generates input/output samples (patterns), which will be used to train the NN. Among the samples, 100 samples are used as the training NN data, while the rest 100 samples are used as the testing NN data. The parameters of TCPSO are applied in NN training set respectively to swarm size $m=20$, acceleration coefficients $c_1=c_2=2$, inertia weight $\omega=0.5$, maximal velocity $v_{\max}=1$. The NN training and the generalization capability of the trained NN test are shown in Fig. 6. From Fig. 6, it can be seen that the NN trained by TCPSO has the accurate prediction capability and good generalization capability, and TCPSO is independent of the initial NN weights.

The nonlinear model of the plant was placed into the NNPC control loop. The initial condition of the plant is $(y(t-1),y(t-2))=(0,0)$, and the input constraint $-3\leq u(t)\leq 3$. Here, the parameters of the TCPSO used in the predictive control are the same as in the NN training, except $v_{\max}=3$. Fig. 7 presents the control results obtained by TCPSO method for square signal and it shows a good tracking result. The TCPSO method is compared with the quasi-Newton, GA in terms of mean-square-tracking error for square and sine signals as shown in Table II. It shows TCPSO method has more small mean-square-tracking error than quasi-Newton and GA methods.

TABLE II
COMPARISON OF THE MEAN-SQUARE-TRACKING ERROR

$r(t)$	quasi-Newton	GA	CPSO	TCPSO
Square	0.2105	0.0870	0.0581	0.0374
Sine	0.0204	0.0129	0.0088	0.0065

VI. Conclusion

In this letter, a new TCPSO is proposed. The TCPSO algorithm combines the fast search ability of PSO with the stochastic and ergodicity property of chaos generated by tent map, which has the outstanding advantages and higher iterative speed than the logistic map. Results of two benchmark tests show that TCPSO is superior to PSO, MPSO, CPSO, and GA in terms of the searching solutions.

Moreover, the TCPSO is used as the optimization technique in NNPC. It can avoid calculating the complex Hessian or Jacobian matrices required in gradient-based methods and reduce the computation effort of NNPC. The simulation results show that TCPSO-based NNPC is very effective.

References

[1] D. W. Clarke, C. Mohtadi, and P. S. Tuffs, "Generalized predictive control—Part I and Part II," *Automatica*, vol. 23, no. 2, pp. 137–160, 1987.

[2] M. A. Henson, "Nonlinear model predictive control: Current status and future directions," *Comp. Chem. Eng.*, vol. 23, no. 2, pp. 187–202, 1998.

[3] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Netw.*, vol. 1, no. 1, pp. 4–27, Jan. 1990.

[4] S. Z. Du, Z. Q. Chen, Z. Z. Yuan, and X. H. Zhang, "Sensitivity to noise in bidirectional associative memory (BAM)," *IEEE Trans. Neural Netw.*, vol. 16, no. 4, pp. 887–898, Jul. 2005.

[5] X. Li, G. R. Chen, Z. Q. Chen, and Z. Z. Yuan, "Chaotifying linear Elman networks," *IEEE Trans. Neural Netw.*, vol. 13, no. 5, pp. 1193–1199, Sep. 2002.

[6] M. A. Hussian, "Review of the applications of neural networks in chemical process control—Simulation and online implementation," *Artif. Intell. Eng.*, vol. 13, no. 1, pp. 55–68, 1999.

[7] J. Saint-Donat, N. Bhat, and T. J. McAvoy, "Neural net based model predictive control," *Int. J. Control*, vol. 54, no. 6, pp. 1453–1468, 1991.

[8] Y. Tan and A. Cauwenberghe, "Nonlinear one-step-ahead control using neural networks: Control strategy and stability design," *Automatica*, vol. 32, no. 12, pp. 1701–1706, 1996.

[9] J. R. Noriega and H. Wang, "A direct adaptive neural-network control for unknown nonlinear systems and its application," *IEEE Trans. Neural Netw.*, vol. 9, no. 1, pp. 27–34, Jan. 1998.

[10] D. Soloway and P. J. Haley, "Neural generalized predictive control: A Newton-Raphson implementation," in *Proc. IEEE Int. Symp. Intell. Control*, Sep. 15–18, 1996, pp. 277–282.

[11] S. C. Shin and S. B. Park, "GA-based predictive control for nonlinear processes," *Electron. Lett.*, vol. 34, no. 20, pp. 1980–1981, 1998.

[12] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, Perth, Australia, Nov. 27–Dec. 1, 1995, vol. 4, pp. 1942–1948.

[13] P. J. Angeline, "Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences," in *Proc. 7th Annu. Conf. Evol. Programming VII*, 1998, pp. 601–610.

[14] M. Clerc and J. Kennedy, "The particle swarm—Explosion, stability, and convergence in a multidimensional complex space," *IEEE Trans. Evol. Comp.*, vol. 6, no. 1, pp. 58–73, Feb. 2002.

[15] A. Stacey, M. Jancie, and I. Grundy, "Particle swarm optimization with mutation," in *Proc. IEEE Congr. Evol. Comp. (CEC)*, 2003, pp. 1425–1430.

[16] B. Li and W. S. Jiang, "Optimizing complex functions by chaos search," *Cybern. Syst.*, vol. 29, no. 4, pp. 409–419, 1998.

[17] X. Xie, W. Zang, and Z. Yang, "A dissipative swarm optimization," in *Proc. IEEE Congr. Evol. Comp. (CEC)*, May 2002, pp. 1456–1461.

[18] M. Iqbal, A. A. Freitas, and C. G. Johnson, E.-G. Talbi, Ed., "Varying the topology and the probability of re-initialization in dissipative particle swarm optimization," *Evol. Artif.*, pp. 1–12, 2005.

[19] B. Liu, L. Wang, and Y. H. Jin, "Improved particle swarm optimization combined with chaos," *Chaos, Solitons Fractals*, vol. 25, no. 5, pp. 1261–1271, 2005.

[20] X. H. Wang and J. M. Xiao, "PSO-based model predictive control for nonlinear processes," in *Lecture Notes in Computer Science*, ser. 3611. Berlin, Germany: Springer-Verlag, 2005, pp. 196–203.

[21] J. P. Coelho, P. B. M. Oliveira, and J. B. Cunha, "Greenhouse air temperature predictive control using the particle swarm optimisation algorithm," *Comp. Electr. Agriculture*, vol. 49, no. 3, pp. 330–344, 2005.

[22] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

Neurocomputing Model for Computation of an Approximate Convex Hull of a Set of Points and Spheres

Srimanta Pal and Sabyasachi Bhattacharya

***Abstract*—In this letter, a two-layer neural network is proposed for computation of an approximate convex hull of a set of given points in 3-D or a set of spheres of different sizes. The algorithm is designed based on an elegant concept—shrinking of a spherical rubber balloon surrounding the set of objects in 3-D. Logically, a set of neurons is orderly placed on a spherical mesh i.e., on a rubber balloon surrounding the objects. Each neuron has a parameter vector associated with its current position. The resultant force of attraction between a neuron and each of the given points/objects, determines the direction of a movement of the neuron lying on the rubber balloon. As the network evolves, the neurons (parameter vectors) approximate the convex hull more and more accurately.**

***Index Terms*—Convex hull, energy function, neural networks.**

I. Introduction

The computation of a *convex hull* of a set of points or objects in 2-D or higher dimensions is a well-known problem. It has been a topic of research for a long time in computational geometry [3], [8], [20]. The convex hull of a given set of points (or circles/ellipses) in a plane is defined as the smallest convex polygon/polycurve containing all the points (or circles/ellipses) in the set. The philosophy behind the computation of the 2-D (or 3-D) convex hull of a set of objects (e.g., points, circles, ellipses, spheres, etc.) can be easily understood with the help of a rubber band (or balloon): stretch a rubber band (or balloon) to surround the set of given objects and then release it to shrink. On equilibrium, the rubber band (or balloon) defines the convex hull.

Manuscript received May 11, 2005; revised October 14, 2005 and September 19, 2006; accepted September 26, 2006.

S. Pal is with the Electronics and Communication Sciences Unit, Indian Statistical Institute, Calcutta 700 108, India (e-mail: srimanta@isical.ac.in).

S. Bhattacharya is with the Computer Consultant Department, TATA Consultancy Services Ltd., Salt Lake, Calcutta 700 091, India (e-mail: sabyasachi.bhattacharya@tcs.com).

Digital Object Identifier 10.1109/TNN.2007.891201

The computation of the convex hull of a finite set of points [1]–[6], [8]–[10], [12], [14], [15], [19], [20], [24], of a polygon [3], [8], [20], or of a finite set of circles [3], [7], [17], [18], [21], particularly in the plane, has been studied and has wide applications in pattern recognition, image processing, cluster analysis, statistics, robust estimation, operations research, computer graphics, robotics, communication, and several other fields. Since the 1970s, the problem of the convex-hull computation has been an interesting area of research. As a result, a number of algorithms are available in the literature to solve this problem. These algorithms can broadly be classified as follows: 1) computing exact or approximate convex hull, 2) using sequential [2], [3], [8], [9], [20], parallel [1], [4], [10], [14], or neural computation [5], [6], [12], [15]–[18], [24], 3) convex hull in the 2-D/3-D or higher dimensions, 4) computing *offline*, *online*, or dynamic processing mode, and 5) for a point set or other object set (such as circle, ellipse, sphere, polygon, Jordan curve, etc.).

The algorithms in [5], [6], [12], [15]–[18], and [24] compute the convex hull of a set of planar points and they are designed using artificial neural networks. Wennmyr [24] proposed an exact convex-hull computation algorithm based on a multilayer perceptron network [13], [22]. He designed a network that can decide whether a given point is inside a convex polygon (using the fact that a convex polygon is always the intersection of half-planes). In this network, every node at the lowest level has a different decision boundary. Leung *et al.* [12] proposed algorithms for computation of an approximate convex hull in online as well as offline modes. The network consists of an input layer and an output layer of neurons. Datta *et al.* [5], [6] and Pal *et al.* [15]–[17] proposed exact convex-hull computation algorithms based on self-organization technique [11] in offline and dynamic [16] modes.

In this letter, we propose a two-layered neural network model for the computation of an approximate convex hull of a set of points in 3-D or a set of spheres of different sizes. This model is basically an extension of the method for the computation of an approximate convex hull of a set of 2-D points, circles, and ellipses [18]. This proposed model is also applicable for different kinds of objects such as ellipsoid or closed regular objects and in 2-D, 3-D, or higher dimensions. Here, the proposed neural network contains two layers: bottom and top layers. The top layer of the network contains the following: 1) n storage nodes which accept input vectors and 2) N composite nodes connected in a mesh on a spherical surface. Again, each composite node contains three computing nodes where each one is connected with all storage nodes within the top layer. Bottom layer also contains a mesh-like network with N computing nodes. The (i,j)th composite node in the top layer is connected to the (i,j)th computing node in the bottom layer. The information from the top layer is a feedback to the bottom layer. Each of the computing nodes in the top layer takes part in the convex-hull computation. The top layer provides the convex hull. Initially, each of the N computing nodes in the top layer or an element of a composite node in the bottom layer contains a vector representing its current coordinate (its position on the surface of a sphere that surrounds the set of given points in 3-D/spheres). Each storage node in the top layer is used to store an input vector [e.g., in case of a point set (x,y,z)-coordinate of a given point and in case of a sphere, the input vector is (x_c, y_c, z_c, r), where (x_c, y_c, z_c) is the center and r is the radius of a given sphere, etc.]. The information stored in a storage node does not change during processing. Computing nodes determine an approximate convex hull from the given input set. The contents of each computing node are updated using a *force of attraction* philosophy. This force of attraction is determined between a computing node and all other storage nodes. The updating maintains the convexity due to a tensile force by its neighboring computing nodes. Initially, the computing nodes are placed on a spherical mesh in an ordered fashion. As the net evolves, the initial mesh forms an approximate convex hull. This approximation depends mainly on two parameters: N the starting number of computing nodes and the step size α (say) of movement along the direction of the resultant force acting on it.

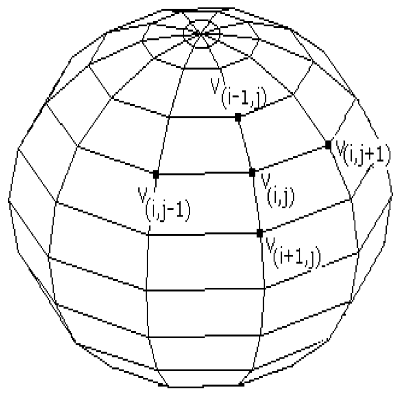

Fig. 1. Arrangements of the dynamic points $V_{i,j}$ on the sphere $C(0)$ to enclose all the points in S.

II. Model

A. Formulation

Consider a set S of n points in 3-D representing the input vectors (the signals)

$$\begin{aligned} S &= \{(x_1,y_1,z_1),(x_2,y_2,z_2),\ldots,(x_n,y_n,z_n)\} \\ &= \{P_1,P_2,\ldots,P_n\} \end{aligned} \tag{1}$$

where P_i is the ith point in S. Let $X_i = (x_i, y_i, z_i)$ be the coordinate of P_i.

The convex hull of the set S is defined as follows.

Definition 1: The *convex hull* of a d-dimensional set S is the smallest convex set containing S. The convex hull here is in fact a convex hyperpolyhedron. Each edge of the hyperpolyhedron is a hull edge and each of its vertices is a hull vertex [3], [8], [20].

Suppose $\{N_c^{(i,j)} | i,j = 0,1,\cdots,N-1\}$ is a set of dynamic points, i.e., computing processors which are placed in an order on a mesh/grid structure of a sphere $C(t)$ (balloon) which encloses the given set of points in S. Initially, the dynamic points $N_c^{(i,j)}$ for $i,j = 0,1,\cdots,N-1$ are on the sphere $C(t)$, such that $N_c^{(i,N)} = N_c^{(i,0)}$ for $i = 0,1,\ldots,N-1$ (Fig. 1). If we take horizontal (or vertical) cross sections of the sphere through any dynamic point we get N points in each horizontal (or vertical) plane. At time t, each processor $N_c^{(i,j)}$ stores a vector $V_{ij}^{(t)} = (x_{ij}(t), y_{ij}(t), z_{ij}(t))$ which is its coordinate for $i,j = 0,1,\cdots,N-1$.

Initially, the sphere $C(0)$ with the radius r_0 surrounds S and the coordinates of the N^2 dynamic points, $N_c^{(i,j)}$ with $V_{ij}^{(0)}$, $i,j = 0,1,\cdots,N-1$, on $C(0)$ are computed using (2)

$$\begin{aligned} x_{ij}(0) &= r_0 \cos\phi_i \cos\theta_j \\ y_{ij}(0) &= r_0 \cos\phi_i \sin\theta_j \\ z_{ij}(0) &= r_0 \sin\phi_i \end{aligned} \tag{2}$$

where

$$\theta_j = \frac{2j-N}{N}.\pi \quad \phi_i = \frac{2i-N}{N}.\frac{\pi}{2} \tag{3}$$

for $i,j = 0,1,\ldots,N-1$.

Each dynamic point $N_c^{(i,j)}$ maintains a weight vector $W_{ij} = (Wx_{ij}, Wy_{ij}, Wz_{ij})$, initialized by the unit vector directed from the center of the sphere $C(0)$ to the point $N_c^{(i,j)}$ with $V_{ij}^{(0)}$. In other