HYBRID SOFT COMPUTING APPROACHES TO IDENTIFICATION OF NONLINEAR SYSTEMS

Shigeyasu Kawaji [*]

[*] *Graduate School of Science and Technology*
Kumamoto University
2-39-1 Kurokami, Kumamoto 860-8555, Japan
kawaji@cs.kumamoto-u.ac.jp

Abstract: *This paper is concerned with the identification of nonlinear systems by utilizing of hybrid soft computing approaches. Based on the flexibly computational structure of the tree, a unified framework is constructed in which various soft computing models can be developed, evolved and evaluated. In this framework, the architecture of the hybrid soft computing models is created and evolved by using the modified probabilistic incremental program evolution (MPIPE) algorithm, and the parameters used in hybrid soft computing models can be optimized using a class of optimization techniques. Simulation results for the identification of nonlinear systems show the feasibility and effectiveness of the proposed method.*

Keywords: Modified Probabilistic Incremental Program Evolution, Hybrid soft computing, Random search, Identification, Nonlinear system

1. INTRODUCTION

Soft computing approaches have been successfully applied to many engineering and scientific fields in recent years, especially for the identification and control of nonlinear systems. These methodologies have shown some advantages in dealing with a number of difficult identification and control problems than the conventional approaches [1][2].

The system identification and controller design problem have been studied under a variety of titles including neural networks [3]-[5], fuzzy systems [6]-[8] and evolutionary computation approaches [9]. The basic idea of these methods is that the nonlinear system identification and control problem can be posed as a nonlinear function approximation problem. Thus, the performance of the identification and controller depends largely on the characteristics of the approximators.

In this paper, based on the flexible computational structure of the tree, a unified framework is constructed in which various soft computing models can be developed, evolved and evaluated. In this framework, the architecture of the hybrid soft computing models is created and evolved using modified probabilistic incremental program evolution (MPIPE) algorithm [10]-[12], and the related parameters are optimized by a combination of least square and random searcch algorithms.

The paper is organized as follows: Section 2 gives a unified framework for evolving the hybrid soft computing models and its learning algorithm. Some simulation results to confirm the feasibility and effectiveness of the proposed methods are presented in Section 3. Finally in section 4 we present some conclusions and future works.

2. HYBRID SOFT COMPUTING: A UNIFIED FRAMEWORK

The basic idea is that if a soft computing model can be represented as a type constrained sparse tree, some advantages may be appeared. First, the different architecture of soft computing models can be created via creating the different trees. And then some tree structure based evolutionary algorithms, i.e., a modified PIPE algorithm can be used to evolve the architecture of the soft computing models.

2.1 *Representation and calculation*

In our previous work [19], multilayer peceptron networks and additive and direct neurofuzzy models have been represented and calculated as type constrained sparse tree, and evolved using MPIPE and random search algorithm.

In what follows, we focus on the tree structural representation of the basis function networks, in which each of the basis function networks can be coded as a type constrained sparse tree. There is no need to encode and decode between the tree and the basis function networks in the calculation of the basis function networks. So, the optimization of the basis function networks can be directly replaced by the evolutionary induction of the type constrained sparse tree.

The node's instruction of the tree in the layer 0, 1 and 2 is selected from the different instruction sets. Three instruction sets I_0, I_1 and I_2 are used in the creation of the tree. The instruction I_0 is used to control the size of hidden layer of the basis function networks or the number of basis functions. The instruction I_1 is used for a selecting the components of the basis functions in VPBF networks and to control the connection ways in other basis function networks. The instruction I_2 is used to select the inputs neurons.

In the following, the type constrained sparse tree (genotype) and its corresponding the basis function networks are given in detail.

2.2 *Volterra Polynomial Basis Function Net*

A network whose basis functions consist of the Volterra polynomials is named as the Volterra polynomial basis function network [15][16]. For function approximation, the usually used Volterra polynomial basis functions is

$$\begin{aligned}\phi &= \{\phi_0(x), \phi_1(x), \ldots, \phi_N(x)\} \\ &= \{1, x_0, x_1, \ldots, x_n, x_0x_1, x_0x_2, \ldots, x_0x_n, x_1x_2, \\ &\quad x_1x_3, \ldots, x_1x_n, \ldots, x_{n-1}x_n, {x_0}^2, \ldots, {x_n}^2\} \quad (1)\end{aligned}$$

where $N = (n+1)(n+2)/2$ is the number of Volterra polynomial basis functions, and n is the number of inputs. Then, the nonlinear function can be represented/approximated as

$$f(x) = \sum_{i=0}^{N} \omega_i * \phi_i(x) + O(x^3) \quad (2)$$

where $O(x^3)$ represents the model mismatch. For complex approximation problems, the higher order Volterra polynomial basis functions may be needed. One of type constrained sparse trees represented as a VPBF-NN is shown in Fig.1. The used instruction sets for generating the tree are $I_0 = \{+_2, \ldots, +_N\}$, $I_1 = \{R, x_0, x_1, \ldots, x_n, *2, *3\}$ and $I_2 = \{x_0, x_1, \ldots, x_n\}$, where instruction $+N$ is used to control the number of Volterra polynomial basis functions for approximating the nonlinear functions at hand, the instruction $*2$ and $*3$ are used to select the higher components of the Volterra polynomial basis function set.

The main problem of optimizing the VPBF-NN is determination of network architecture and estimation of the corresponding parameter for each Volterra polynomial function.

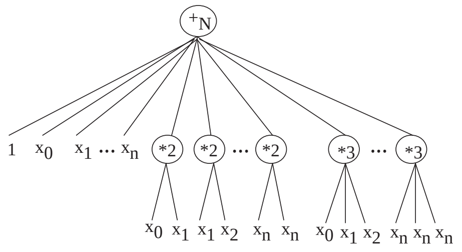

Fig. 1. A tree structural representation of VPBF network

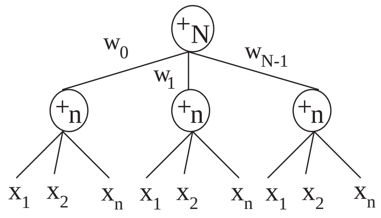

Fig. 2. A type constrained sparse tree (genotype)

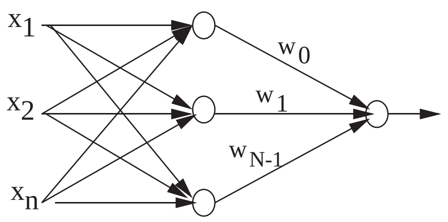

Fig. 3. Corresponding basis function network of the type constrained sparse tree (phenotype) in Fig.2

2.3 *GRBF, B-spline, Wavelet, Fuzzy Basis, Recurrent Fuzzy Neural, and Local linear GRBF Net and Trees*

One of the type constrained sparse tree and its corresponding GRBF (B-spline, wavelet, fuzzy, recurrent fuzzy neural and local GRBF basis function) network is shown in Fig.2 and Fig.3, respectively. The used instruction sets for generating the tree are $I_0 = \{+_2, \ldots, +_N\}$, $I_1 = \{+_n\}$ and $I_2 = \{x_1, x_2, \ldots, x_n\}$. N is used to control the numbers of hidden neurons which is also the number of basis functions in the hidden layer of the basis function networks, and n is the number of inputs of the basis function networks.

- *GRBF Network*

The used Gaussian radial basis function is given by

$$\phi_i(x) = exp(-\frac{\| x - c_i \|^2}{{d_i}^2}) \quad (3)$$

where $\phi_i(x)$ is i-th Gaussian radial basis function, $x \in R^n$ is input vector, c_i and d_i are the center and width of i-th basis function. Thus, the output of whole tree in the Fig.2 is calculated as

$$y = \sum_{i=0}^{N-1} \omega_i * exp(-\frac{\| x - c_i \|^2}{{d_i}^2}) \quad (4)$$

The objective of optimization of GRBF networks is determination of the number of basis functions, the center and width of each Gaussian radial basis functions by appropriate methods.

• *B-spline Network*

The B-spline functions can be defined in a recursive way as

$$B_0(t) = \begin{cases} 1, \ t \in [-\frac{1}{2}, \frac{1}{2}] \\ 0, \ otherwise \end{cases} \quad (5)$$

$$B_m(t) = (B_{m-1} * B_0)(t) \quad (6)$$

Then, the translation and dilation of B-spline function of order 2 is given by

$$f_{a,b}(t) = B_2(\frac{t-b}{a}) \quad (7)$$

For the n-dimensional input space $x = [x_0, x_1, \dots, x_{n-1}]$, the multivariate B-spline functions is selected by the product of single B-spline functions in (7) as

$$N_i(x) = \prod_{j=0}^{n-1} B_2(\frac{x_j - b_j}{a_j}) \quad (8)$$

Thus, the output of whole tree in the Fig.2 is calculated as

$$y = \sum_{i=0}^{N-1} \omega_i * N_i(x) = \sum_{i=0}^{N-1} \omega_i * \prod_{j=0}^{n-1} B_2(\frac{x_j - b_j}{a_j}) (9)$$

The objective of optimization of B-spline networks is determination of the number of basis functions, the center and width of each B-spline basis functions. It can be seen that in contrast to the usually methods the pre-partition of input space is not needed in our approach.

• *Wavelet Neural Network*

Given the mother wavelet, a family of equally shaped functions by shifts in time (translation) and scaling (dilation) can be obtained as

$$\psi_{a,b}(t) = \frac{1}{\sqrt{|a|}} \psi(\frac{t-b}{a}), a \neq 0, a, b \in R \quad (10)$$

For the n-dimensional input space $x = [x_1, x_2, \dots, x_n]$, the multivariate wavelet basis function is calculated by the product of n single wavelet basis functions in (10) as

$$s_i(x) = \prod_{j=1}^{n} \frac{1}{\sqrt{|a_j|}} \psi(\frac{x_j - b_j}{a_j}) \quad (11)$$

Thus, the overall output of the wavelet basis function network is calculated as

$$y = \sum_{i=0}^{N-1} \omega_i * s_i(x) = \sum_{i=0}^{N-1} \omega_i * \prod_{j=0}^{n-1} \frac{1}{\sqrt{|a_j|}} \psi(\frac{x_j - b_j}{a_j}) \quad (12)$$

The problem in designing the wavelet basis function network is to determine the optimal network size (the numbers of the wavelet basis functions), the parameter a_j and b_j ($j = 0, 1, \dots, (N-1)*(n-1)$) for each single wavelet basis function. It can be seen that in contrast to the usually methods the pre-determining of the parameter a_j and b_j is not needed in our approach.

• *Fuzzy basis function Networks*

In adaptive fuzzy systems, the fuzzy rule is represented as (T-S Model)

if x_0 is A_{i0} and x_1 is A_{i1} ... and x_{n-1} is $A_{i(n-1)}$,
then $y_i = b_{i0}x_0 + b_{i1}x_1 + ... + b_{i(n-1)}x_{n-1} + b_{in}$
(i=0, 1, ... N-1)

When algebraic operators are used to implement fuzzy logic functions, the real valued inputs are represented via fuzzy membership function, and a center of defuzzification method is used, then the output of fuzzy basis function network is given by

$$y = \sum_{i=0}^{N-1} \phi_i(x) * \omega_i \quad (13)$$

$$\phi_i(x) = \frac{\prod_{j=0}^{n-1} \mu_{A_{ij}}(x_j)}{\sum_{i=0}^{N-1} \prod_{j=0}^{n-1} \mu_{A_{ij}}(x_j)} \quad (14)$$

Thus, the output of the fuzzy basis function networks or corresponding tree is calculated as

$$y = \sum_{i=0}^{N-1} \omega_i * \frac{\prod_{j=0}^{n-1} \mu_{A_{ij}}(x_j)}{\sum_{i=0}^{N-1} \prod_{j=0}^{n-1} \mu_{A_{ij}}(x_j)} \quad (15)$$

The objective is that directly learning and evolving the number of fuzzy rules, determining the parameters of each fuzzy membership functions, and optimizing the corresponding weight of each fuzzy basis functions for fitting a given data set. In our experiments of this research the selected fuzzy membership function is Cauchy function. It is also valuable to mention that the initial fuzzy partition of input space is not needed in our approaches.

• *Recurrent Fuzzy Neural Network*

In RFNN, an extention of usual FNN the feedback connections are added in the second layer of FNN, and The input/output representation of RFNN

$$y(k) = \sum_{j=1}^{m} \omega_{mj} \prod_{i=1}^{n} exp(-\frac{(x_i(k) + O_{ij}^2(k-1) \cdot \theta_{ij} - m_{ij})^2}{(\sigma_{ij})^2}) \quad (16)$$

$$O_{ij}^2(k-1) = exp(-\frac{(x(k-1) + O_{ij}^2(k-2) \cdot \theta_{ij} - m_{ij})^2}{(\sigma_{ij})^2}) \quad (17)$$

where n is the number of input variables, and m is the number of term nodes for each input variable. The used membership function is the Gaussian function as,

$$\mu_{ij} = exp(-\frac{(x_{ij} - m_{ij})^2}{(\sigma_{ij})^2}) \quad (18)$$

where m_{ij} and σ_{ij} are the center and the width of Gaussian membership function. The subscript ij indicates the j-th term of the i-th input x_i.

- *Local Linear Gaussian Basis Function Network*

Local linear Gaussian basis function network is an extended radial function network which obtained by replacing the output layer weights with linear function of the network inputs. Each neuron represents a local linear model with its corresponding validity function. Furthermore, the radial basis function network is normalized, *i.e.*, the sum of all validity functions for a specific input combination sums up to one. The Gaussian validity functions determine the regions of the input space where each neuron is active. The input space of the net is divided into N hyper-rectangles each represented by a linear function.

The output of a local linear Gaussian basis function network with n inputs x_1,x_2,..., x_n is calculated by summing up the contributions of all N local linear models

$$y = \sum_{i=1}^{N} (\omega_{i0} + \omega_{i1}x_1 + \ldots + \omega_{in}x_n)\phi_i(x) \quad (19)$$

where ω_{ij} are the parameters of the ith linear regression model and x_i is the model input. The validity functions ϕ_i are typically chosen as normalized Gaussian weighting function:

$$\phi_i(x) = \frac{\mu_i}{\sum_{j=1}^{N} \mu_j} \quad (20)$$

$$: \mu_i = exp(-\frac{1}{2}\frac{(x_1 - c_{i1})^2}{\sigma_{i1}^2} - \ldots - \frac{1}{2}\frac{(x_n - c_{in})^2}{\sigma_{in}^2})$$

2.4 *The Proposed Learning Algorithm*

2.4.1. PIPE and MPIPE

PIPE is a recent discrete method for automated program synthesis[12], which contains probability vector coding of program instructions, population-based incremental learning and tree-coded programs like those used in variants of genetic programming. The main principle of PIPE algorithm is that it increases the probability of the best program to be found by using adaptive tuning of the probability distribution for choosing the proper instructions.

In order to construct a unified framework of hybrid soft computing models, PIPE is modified as follows: (1) the sparse tree is constrained by using different instruction sets; (2) the specified data structure is added to the node of the tree for dealing with the parameter learning problem; (3) the initial probability of selecting instructions and the mutation probability are modified accordingly.

2.4.2. Structure Optimization

The probability of selecting instruction in the instruction set I_0, I_1 and I_2 are initialized as

$$P\left(I_{d,w}\right) = \frac{1}{l_i}, \forall I_{d,w} \in I_i, i = 0, 1, 2 \quad (21)$$

where $I_{d,w}$ denotes the instruction of the node with depth d and width w, l_i is the number of instructions in the instruction set I_i. Then, the learning procedure for structure optimization can be summarized as follows:

1) Initially a population of the tree (include the parameters attached to the nodes) is randomly generated according to the predefined the probability of selecting instructions.

2) Let P_{ROG_b} and $P_{ROG_{el}}$ be the best program of the current generation (best program) and the one found so far (elitist program), respectively. Define the probability and the target probability of best program as

$$P\left(P_{ROG_b}\right) = \prod_{\substack{I_{d,w}: used\ to \\ generate\ P_{ROG_b}}} P\left(I_{d,w}\right) \quad (22)$$

and

$$P_{TARGET} = P(P_{ROG_b})$$

$$+(1 - P(P_{ROG_b}))\frac{\varepsilon + FIT(P_{ROGel})}{\varepsilon + FIT(P_{ROG_b})} \quad (23)$$

where $FIT\left(P_{ROG_b}\right)$ and $FIT\left(P_{ROGel}\right)$ denote the fitnesses of the best and elitist programs. In order to increase the probability $P(P_{ROG_b})$, the following process is repeated until $P\left(P_{ROG_b}\right) \geq P_{TARGET}$:

$$P\left(I_{d,w}\right) = P\left(I_{d,w}\right) + c^{lr} \cdot lr \cdot \left(1 - P\left(I_{d,w}\right)\right) (24)$$

where c^{lr} is a constant influencing the number of iterations and ε is the fitness constant.

3) Define the mutation probability as

$$P_{M_p} = \frac{P_M}{(l_0 + l_1 + l_2) \cdot \sqrt{|P_{ROG_b}|}} \quad (25)$$

where $|P_{ROG_b}|$ denotes the number of nodes in program. All the probabilities $P\left(I_{d,w}\right)$ are mutated with probability P_{M_P} according to

$$P\left(I_{d,w}\right) = P\left(I_{d,w}\right) + mr \cdot \left(1 - P\left(I_{d,w}\right)\right) (26)$$

4) Repeat this process according to the new probabilities of selecting instructions in the instruction sets until the best structure found or the maximum number of MPIPE algorithm is reached.

2.4.3. Parameter Optimization

A number of parameter tuning strategies, i.e., genetic algorithms, gradient descent methods, evolutionary programming and random search algorithm can be used to adjust the parameters used in hybrid soft computing models. In this research, the linear-in-parameters are optimized by least square and the nonlinear-in-parameters are optimized by a random search algorithm [17].

2.4.4. The Proposed Algorithm

The proposed hybrid algorithm can be summarized as follows:

1) Set the initial values of parameters used in the PIPE and random search. Set the elitist program as NULL and its fitness value as a biggest positive real number of the computer at hand. Create the initial population (tree) and corresponding weights, parameters used in the basis functions.
2) Structure optimization by PIPE algorithm, in which the fitness function is calculated by Mean Square Error (MSE)
$$Fit(i) = \frac{1}{P-1}\sum_{j=0}^{P}(y_1^j - y_2^j)^2 \qquad (27)$$
where P is the total number of samples, y_1^i and y_2^i are the actual and model output of i-th sample $Fit(i)$ denotes the fitness value of i-th individual.
3) If the better structure found, then go to step 4), otherwise go to step 2). The criterion concerning with better structure found is distinguished as follows: if the fitness value of the best program is smaller than that of the elitist program, or the fitness values of two programs are equal but the nodes of the former is lower than the later, then the better structure is found.
4) Parameter optimization, in which the parameter vector $W(k)$ of best program (tree) is taken out from the population and is optimized in order to decrease the fitness value of best program.
5) If the maximum number of random search is reached, then go to step 6); otherwise go to step 4).
6) If a satisfied solution is found, then stop; otherwise go to step 2).

3. EXPERIMENTS

We present some simulation results to verify the effectiveness of the proposed method for identification of nonlinear system.

The used parameters in MPIPE is shown in Table 1. And the initial values of parameters in random search are listed as

$\beta_0 = 0.1$, $\beta_1 = 1000$, $\alpha = 0.995$, $\phi_0 = 0.1$, $\phi_{min} = 0.001$, $I_{sf0} = 10$, $P_{sf0} = 0.3$, $\triangle I_{sf1} = 0.02$, $\triangle I_{sf2} = 0.1$, $I_{sfmax} = 100$, $K_{er} = 1.001$, $c_i = 1.01$, and $c_d = 0.995$.

Table 1. Parameters of MPIPE Algorithm

Population Size PS	10
Elitist Learning Probability P_{el}	0.01
Learning Rate lr	0.01
Fitness Constant ε	0.000001
Overall Mutation Probability P_M	0.4
Mutation Rate mr	0.4
Prune Threshold T_P	0.999999
Initial Weights	[-0.5, 0.5]

Example 1

The first plant to be identified is a benchmark nonlinear autoregressive time series

$$\begin{aligned} y(k) = &(0.8 - 0.5exp(-y^2(k-1)))y(k-1) \\ &-(0.3 + 0.9exp(-y^2(k-1)))y(k-2) \\ &+0.1sin(3.1415926y(k-1)) + e(k) \quad (28)\end{aligned}$$

where the noise $e(k)$ was a gaussian white sequence with mean zero and variance 0.02. Among 600 data points generated, the first 500 points were used as an estimation data set and 100 data as a validation data set. The input vector is set as $x = [y(k-1), y(k-2)]$.

Example 2

The second plant to be identified is given by the following equation [5]

$$y(k+1) = \frac{y(k)y(k-1)y(k-2)u(k-1)(y(k-2)-1) + u(k)}{1 + y^2(k-1) + y^2(k-2)} \qquad (29)$$

The input and output of system are $x(k) = [u(k), u(k-1), y(k), y(k-1), y(k-2)]$ and $y(k+1)$, respectively.

The training samples and the test data set are generated by using input signal of (30) and (31), respectively.

$$u(k) = \begin{cases} sin\left(\frac{2\pi k}{250}\right), & if\ (k < 500) \\ 0.8sin\left(\frac{2\pi k}{250}\right) + 0.2sin\left(\frac{2\pi k}{25}\right), & \\ \qquad if\ (k >= 500) & \end{cases} \qquad (30)$$

$$\begin{aligned} u(k) = &0.3sin(k\pi/25) + 0.1sin(k\pi/32) \\ &+0.1sin(k\pi/10) \qquad (31)\end{aligned}$$

The simulation results for Examples 1 and 2 are shown in Table 2. From the simulation results, it can be seen that the proposed method works very well for generating the different basis function networks. For the comparison, the VPBF network in general has smaller number of parameters, good approximation ability and fast convergence speed for current examples. In general, the evolved GBRF, B-spline and fuzzy basis function networks

Table 2. The number of parameters, the training and test error of the evolved basis function networks

Experiment	No. of para.	MSE for training	MSE for validation
Ex. 1 VPBF	7	0.000587	0.000596
Ex. 1 GRBF	72	0.000602	0.000653
Ex. 1 B-spline	80	0.000523	0.000618
Ex. 1 Wavelet	75	0.000471	0.000537
Ex. 1 FNN	80	0.000662	0.000570
Ex. 1 RFNN	96	0.000362	0.000371
Ex. 1 LRBF	96	0.000427	0.000450
Ex. 2 VPBF	13	0.000041	0.000071
Ex. 2 GRBF	77	0.000151	0.000126
Ex. 2 B-spline	209	0.000133	0.000097
Ex. 2 Wavelet	209	0.000179	0.000121
Ex. 2 FNN	176	0.000141	0.000065
Ex. 2 RFNN	156	0.000113	0.000055
Ex. 2 LRBF	149	0.000124	0.000087

have smaller size and parameters than the conventional approaches.

4. CONCLUSION

Based on the flexibly computational structure of tree, a unified framework for evolving the basis function networks has been proposed in this paper. Simulation results show that the evolved basis function networks are effective for the identification of nonlinear systems.

The study has shown that (1) it is possible and effective to construct a unified soft computing model in which it is important to find a proper representation of the soft computing models, and (2) it is difficult yet to make a banlance bewteen the structure and parameter learning. These forms our future research interestings.

5. REFERENCE

[1] J. Sjoberg et al. (1995). Nonlinear black - box modeling in system identification: a unified overview. *Automatica*. Vol.31. pp.1691-1724.

[2] S. Kawaji and Y. Chen. (2000). Soft Computing Approach to Nonlinear System Identification. *Proc. of the IEEE Int. Conf. on Industrial Electronics, Control and Instrumentation*. Nagoya. pp.1803-1808.

[3] S.A. Billings (1992). Neural networks and system identification. In K. Warwick et al. (Eds). *Neural networks and system control*. Peter Peregrinus. London. pp.181-205.

[4] S. Chen et al. (1990). Nonlinear system identification using neural networks. *Int. Journal of Control*. Vol.51. No.6. pp.1191-1214.

[5] K. S. Narendra et al.(1990). Identification and Control of Dynamic System using Neural Networks. *IEEE Trans. on Neural Networks*. Vol.1. No.2. pp.4-27.

[6] T. Takagi et al. (1985). Fuzzy identification of systems and its application to modeling and control. *IEEE Trans. on Syst., Man and Cybern.*. Vol.15. pp.116-132.

[7] C. Xu et al. (1987). Fuzzy model identification and self learning for dynamic systems. *IEEE Trans. on Syst. Man and Cybern.*. Vol.17. pp.683-689.

[8] J.S. Jang et al. (1997), Neuro-fuzzy and soft computing: a computational approach to learning and machine intelligence. Prentice-Hall.

[9] K. Kristinn et al. (1992). System identification and control using genetic algorithms. *IEEE Trans. on Systems, Man and Cybern.*. Vol.22. No.5. pp.1033-1046.

[10] Y. Chen and S. Kawaji. (1999). Identification and Control of Nonlinear System using PIPE Algorithm. *Proc. of Workshop on Soft Computing in Industry'99*. Muroran. pp.60-65.

[11] Y. Chen and S. Kawaji. (1999). Evolutionary Control of Discrete-Time Nonlinear System using PIPE Algorithm. *Proc. of IEEE Int. Conf. on Systems, Man and Cybernetics*. Tokyo. pp.1078-1083.

[12] S. Rafal and S. Jurgen. (1997). Probabilistic Incremental Program Evolution. *Evolutionary Computation*. Vol.5. No.2. pp.123-141.

[13] Y. Chen and S. Kawaji. (2000). Evolving Artificial Neural Networks by hybrid approaches of PIPE and Random Search Algorithm. *Proc. of The Third Asian Control Conference*. Shanghai. pp.2206-2211.

[14] Y. Chen and S. Kawaji. (2001). Evolving Neurofuzzy Systems for System Identification. *Proc. of International Symposium on Artificial Life and Robotics*. Tokyo. pp.204-207.

[15] G.P. Liu et al. (1998). On-line identification of nonlinear systems using Volterra polynomial basis function neural networks. *Journal of Neural Networks*. Vol.11. pp.1645-1657.

[16] G.P.Liu et al. (1999). Multiobjective criteria for neural network structure selection and identification of nonlinear systems using genetic algorithms. *IEE Proc.-Control Theory Appl.*. Vol.146. No.5. pp.373-382.

[17] J. Hu, et. al. (1998). RasID - Random Search for Neural Network Training. *Journal of Advanced Computational Intelligence*. Vol.2. No.2. pp.134-141.

[18] S.S. Ge, et al. (1999). Adaptive neural network control of nonlinear systems by state and output feedback. *IEEE Trans. Syst., Man, and Cybern.*. Vol.29. No.6. pp.818-828.

[19] S. Kawaji et al. (2001). Design of Additive Models using Hybrid Soft Computing Approaches *Proc. of IEEE Int. Conf. on Systems, Man, and Cybernetics*. Tucson. pp.1415-1423.