Programing Hierarchical TS Fuzzy Systems

Yuehui Chen, Lizhi Peng
School of Information Science and Engineering
Jinan University
Jinan 250022, Shandong, P.R.China
yhchen@ujn.edu.cn

Ajith Abraham
School of Computer Science and Engineering
Chung-Ang University
Seoul, Republic Korea
ajith.abraham@ieee.org

***Abstract*— In this paper, we focus on an evolutionary algorithm to design hierarchical or multilevel fuzzy system (architecture and parameters) automatically. This research work presents an automatic way of evolving hierarchical Takagi-Sugeno Fuzzy Systems (TS-FS). The hierarchical structure is evolved using Probabilistic Incremental Program Evolution (PIPE) with specific instructions. The fine tuning of the *if-then* rule's parameters encoded in the structure is accomplished using Particle Swarm Optimization (PSO). The proposed method interleaves both PIPE and PSO optimizations. Except for the randomly initialized hierarchical structure and parameters, we further explore the embedding of a clustering algorithm to speed up the learning algorithm. The new method results in a smaller rule-base and good learning ability. The proposed hierarchical TS-FS is evaluated by using Mackey-Glass chaotic time-series forecasting problem. When compared to other hierarchical TS-FS the proposed approach exhibits competing results with high accuracy and smaller size of hierarchical architecture.**

I. Introduction

Fuzzy systems have been successfully used in most applications, such as function approximation, nonlinear system control, time-series prediction, fuzzy classification and fuzzy clustering. Standard fuzzy systems are general applicable, in the sense that they are universal approximators which can approximate arbitrary continuous functions to any accuracy [1],[2]. However, as fuzzy systems have been applied to more complicated and high dimensional systems, the "curse of dimensionality" has become increasingly apparent as the bottleneck to wider application [3].

Hierarchical and/or multi-level fuzzy system is an active research area in recent years due to its nature of rule base reduction and good approximation abilities. Usually, it is difficult to arrange appropriate hierarchical layer and input variables to each sub-fuzzy system for a given problem.

Hierarchical fuzzy systems have attracted considerable attentions in recent years [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16] [22]. Torra [17] has summarized the related recent research work in this domain. As a way to overcome the curse-of-dimensionality, it was suggested by Brown et al. [13] to arrange several low-dimensional rule bases in a hierarchical structure, i.e., a tree, causing the number of possible rules to grow in a linear way according to the number of inputs. A method was proposed to determine automatically the fuzzy rules in a hierarchical fuzzy model [18]. Rainer [11] described a new algorithm which derives the rules for hierarchical fuzzy associative memories that were structured as a binary tree. Wang and Wei [7], [8], [19] proposed specific hierarchical fuzzy systems and its universal approximation property was proved. The approximation capabilities of hierarchical fuzzy systems was further analyzed by Zeng and Keane [20]. But the main problem lies in fact that this is a specific hierarchical fuzzy system which lacks flexibility in structure adaptation, and it is difficult to arrange the input variables for each sub-model. Lin and Lee [21] proposed a genetic algorithm based approach to optimize the hierarchical structure and the parameters of 5-inputs hierarchical fuzzy controller for the low-speed control problem. Based on the analysis of importance of each input variable and the coupling between any two input variables, the problem of how to distribute the input variables to different (levels of) relational modules for incremental and aggregated hierarchical fuzzy relational systems was addressed [14].

In this paper, we focus on an evolutionary procedure to design hierarchical or multilevel fuzzy system (architecture and parameters) automatically. This research work presents an automatic way of evolving hierarchical Takagi-Sugeno Fuzzy Systems (TS-FS). The hierarchical structure is evolved using Probabilistic Incremental Program Evolution (PIPE) with specific instructions. The fine tuning of the *if-then* rule's parameters encoded in the structure is accomplished using Particle Swarm Optimization (PSO). The proposed method interleaves both PIPE and PSO optimizations. Starting with random structures and rules' parameters, it first tries to improve the hierarchical structure and then as soon as an improved structure is found, it further fine tunes the rules' parameters. It then goes back to improve the structure again and, provided it finds a better structure, it again fine tunes the rules' parameters. This loop continues until a satisfactory solution (hierarchical TS-FS model) is found or a time limit is reached. Except for the randomly initialized hierarchical structure and parameters, we further explore the embedding of a clustering algorithm to speed up the learning algorithm. The new method results in a smaller rule-base and good learning ability. This is also the difference between this paper and ref. [23].

The rest of the paper is organized as follows. A simple introduction of Takagi-Sugeno Fuzzy Inference System (TS-FS) is given in Section 2. A new encoding method an automatic design method for the hierarchical TS-FS is presented in Section 3. Some simulation results and discussions related

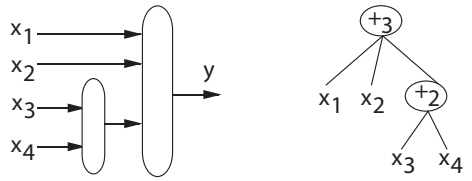

Fig. 1. Left: An example of possible hierarchical TS-FS model with 4 inputs and 3 hierarchical ayers, Right: the tree structural representation of the corresponding hierarchical TS-FS model, where the used instruction set is $I = \{+_2, +_3, x_1, x_2, x_3, x_4\}$.

to time-series prediction problem are provided in Sections 4. Finally in Section 5, we present some conclusions and future works.

II. Takagi-Sugeno Fuzzy Inference System (TS-FS)

Fuzzy inference systems are composed of a set of *if-then* rules. A Takagi-Sugeno fuzzy model has the following form of fuzzy rules [4] :

R_j : if x_1 is A_{1j} and x_2 is A_{2j} and ... and x_n is A_{nj}
Then $y = g_j(x_1, x_2, \ldots, x_n)$, $(j = 1, 2, \ldots, N)$

where $g_j(\cdot)$ is a crisp function of x_i. Usually, $g_j(x_1, x_2, \ldots, x_n) = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \ldots + \omega_n x_n$. The overall output of the fuzzy model can be obtained by:

$$y = \frac{\sum_{j=1}^{N} g_j(\cdot) T_{i=1}^{m_j} \mu_{ij}(x_i)}{\sum_{j=1}^{N} T_{i=1}^{m_j} \mu_{ij}(x_i)} \tag{1}$$

where $1 \leq m_j \leq n$ is the number of input variables that appear in the rule premise, N is the number of fuzzy rules, n is the number of inputs, μ_{ij} is the membership function for fuzzy set, A_{ij} and T is a T-norm for fuzzy conjunction. In this paper, the used fuzzy membership function is

$$\mu(a, b; x) = \frac{1}{1 + (\frac{x-a}{b})^2}. \tag{2}$$

The TS-FS is a single-stage fuzzy system. It is important to partition the input space using some clustering, grid partitioning etc. [24]. The shapes of membership functions in the antecedent parts, and the free parameters in the consequent parts are also to be determined using some adaptive techniques [25], [26].

III. Evolutionary design of hierarchical TS-FS

In this section, an automatic design method of hierarchical TS-FS is presented. The hierarchical structure is created and optimized using PIPE with specific instructions and the fine turning of the rule's parameters encoded in the structure is accomplished using PSO.

A. Encode

A tree-structural based encoding method with specific instruction set is selected for representing a hierarchical TS-FS in this research. The reasons for choosing this representation are that (1) the trees have a natural and typical hierarchical layer; (2) with pre-defined instruction sets, the tree can be created and evolved using the existing tree-structure-based approaches, i.e., Genetic Programming (GP) and PIPE algorithms.

Assume that the used instruction set is $I = \{+_2, +_3, x_1, x_2, x_3, x_4\}$, where $+_2$ and $+_3$ denote non-leaf nodes' instructions and taking 2 and 3 arguments, respectively. x_1,x_2,x_3,x_4 are leaf nodes' instructions and taking zero argument each. In addition, the output of each non-leaf node is calculated as a single T-S fuzzy sub-model. For this reason the non-leaf node $+_2$ is also called a two-inputs T-S fuzzy instruction/operator. Fig.1(right) shows tree structural representation of the hierarchical TS-FS model shown in Fig.1(left).

B. Evaluation

We describe an illustrative example through the Fig.1(right) to show how the hierarchical TS-FS tree is calculated.

First, the output of the TS fuzzy sub-model (node $+2$) is computed. The rule base and outputs of this fuzzy TS sub-model can be obtained by k-means cluster directly. Second, the overall output of the hierarchical TS fuzzy model is computed. It has three inputs, x_1, x_2 and y, the output of the TS fuzzy sub-model (node $+_2$). In this case, the node $+_3$ has two leaf nodes and one non-leaf node. For the leaf nodes, we still implement k-means cluster to obtain their fuzzy rule base according to the data given. Suppose for nodes x_1 and x_2, 2 cluster centers are obtained. For the non-leaf node, assume that the number of fuzzy membership functions for variables y is 2. Then the number of rules is 4 and the general output of the node $+_3$ can be calculated as an usual TS fuzzy model.

It should be noted that the number of rules is determined by the number of cluster centers of leaf nodes and the number of fuzzy sets of non-leaf nodes. The free parameters to be optimized by PSO are the parameters of the non-leaf node's fuzzy membership function parameter and the linear weights in the consequent parts of the rules.

C. PIPE

PIPE [27], [28], [29] combines probability vector coding of program instructions, population based incremental learning [30] and tree-coded programs [31]. PIPE iteratively generates successive populations of functional programs according to an adaptive probability distribution, represented as a Probabilistic Prototype Tree (PPT), over all possible programs. Each iteration uses the best program to refine the distribution. Thus, the structures of promising individuals are learned and encoded in PPT.

1) Instructions and Programs: In PIPE, programs are made of instructions from an instruction set $S = \{I_1, I_2, \ldots, I_n\}$ with n instructions. Instructions are user-defined and problem dependent. Each instruction is either a function or a terminal. Instruction set S therefore consists of a function set $F = \{f_1, f_2, \ldots, f_k\}$ with k functions and a terminal set $T = \{t_1, t_2, \ldots, t_l\}$ with l terminals, where $n = k + l$ holds.

Programs are encoded in n-ary trees, with n being the maximal number of function arguments. Each non-leaf node encodes a function from F and each leaf node a terminal from T. The number of subtrees each node has corresponds to the number of arguments of its function. Each argument is calculated by a subtree. The trees are parsed depth first from left to right.

2) PPT: The PPT stores the knowledge gained from experiences with programs and guides the evolutionary search. It holds random constants and the probability distribution over all possible programs that can be constructed from a predefined instruction set. The PPT is generally a complete n-ary tree with infinitely many nodes, where n is the maximal number of function arguments.

Each node N_j in PPT, with $j > 0$ contains a random constant R_j and a variable probability vector $\overrightarrow{P_j}$. Each $\overrightarrow{P_j}$ has n components, where n is the number of instructions in instruction set S. Each component $P_j(I)$ of $\overrightarrow{P_j}$ denotes the probability of choosing instruction $I \in S$ at node N_j. Each vector $\overrightarrow{P_j}$ is initialized as follows:

$$P_j(I) = \frac{P_T}{l}, \forall I : I \in T \tag{3}$$

$$P_j(I) = \frac{1 - P_T}{k}, \forall I : I \in F, \tag{4}$$

where l is the total number of terminals in T, k is the total number of functions in F, and P_T is initially user-defined constant probability for selecting an instruction from T.

3) Program Generation, Growing and Pruning: Programs are generated according to the probability distribution stored in the PPT. To generate a program P_{ROG} from PPT, an instruction $I \in S$ is selected with probability $P_j(I)$ for each accessed node N_j of PPT. Nodes are accessed in a depth-first way, starting at the root node and traversing PPT from left to right.

A complete PPT is infinite, and each PPT node holds a probability for each instruction, a random constant, and n pointers to following nodes, where n is PPT's arity. Therefore, a large PPT is memory intensive. To reduce memory requirements, it is thus possible to incrementally grow and prune the PPT.

On one hand, it is useful to grow the PPT on demand in order to create a variety of programs. Initially the PPT contains only the root node. Additional nodes are created with each program that accesses non-existing nodes during its generation. On the other hand, apart from reducing memory requirements, pruning also helps to discard the elements of probability distribution that have become irrelevant over time. PPT subtrees attached to nodes that contain at least one probability vector component above a threshold T_P can be pruned. If T_P is set to a sufficiently high value (e.g., $T_P = 0.99999$) only parts of the PPT will be pruned that have a very low probability of being accessed. In case of functions, only those subtrees should be pruned that are not required as function arguments. Figure 3 illustrates the relation between the prototype tree and a possible program tree.

4) Fitness Functions: Similar to the other evolutionary algorithms, PIPE uses a problem-dependent and user-defied fitness function. A fitness function maps programs to scalar, real-valued fitness values that reflect the programs' performances on a given task. Firstly PIPE's fitness functions should be seen as error measures, i.e., mean square error (MSE) or root mean square error (RMSE). A secondary non-user-defined objective for which PIPE always optimizes programs is the program size as measured by number of nodes. Among programs with equal fitness smaller ones are always preferred. This objective constitutes PIPE's built-in Occam's razor.

5) Learning Algorithm: PIPE combines two forms of learning: Generation-Based Learning (GBL) and Elitist Learning (EL). GBL is PIPE's main learning algorithm. EL's purpose is to make the best program found so far as an attractor. PIPE executes:

GBL
REPEAT
    with probability P_{el} DO EL
    otherwise DO GBL
UNTIL termination criterion is reached

Here P_{el} is a user-defined constant in the interval [0,1].

Generation-Based Learning

Step 1. **Creation of Program Population.** A population of programs P_{ROG_j} ($0 < j \leq PS$; PS is population size) is generated using the prototype tree PPT, as described in Section III.A. The PPT is grown on demand.

Step 2. **Population Evaluation.** Each program P_{ROG_j} of the current population is evaluated on the given task and assigned a fitness value $FIT(P_{ROG_j})$ according to the predefined fitness function. The best program of the current population (the one with the smallest fitness value) is denoted P_{ROG_b}. The best program found so far (elitist) is preserved in ${P_{ROG}}^{el}$.

Step 3. **Learning from Population.** Prototype tree probabilities are modified such that the probability $P(P_{ROG_b})$ of creating P_{ROG_b} increases. This procedure is called adapt PPT towards(Progb). This is implemented as follows. First $P(P_{ROG_b})$ is computed by looking at all PPT nodes N_j used to generate P_{ROG_b}:

$$P\left(P_{ROG_b}\right) = \prod_{j:N_j \ used \ to \ generate \ P_{ROG_b}} P_j\left(I_j(P_{ROG_b})\right) \tag{5}$$

where $I_j(P_{ROG_b})$ denotes the instruction of program P_{ROG_b} at node position j. Then a target probability P_{TARGET} for P_{ROG_b} is calculated:

$$P_{TARGET} = P(P_{ROG_b}) + (1 - P(P_{ROG_b})) \cdot lr \cdot \frac{\varepsilon + FIT({P_{ROG}}^{el})}{\varepsilon + FIT(P_{ROG_b})} \tag{6}$$

here lr is a constant learning rate and ε a positive user-defined constant. Given P_{TARGET}, all single node probabilities $P_j(I_j(P_{ROG_b}))$ are increased iteratively:

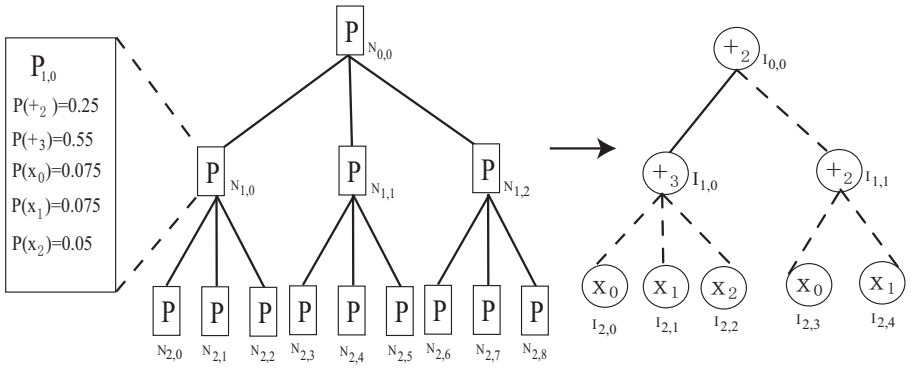

Fig. 2. Example of node $N_{1,0}$'s instruction probability vector $P_{1,0}$ (left). Probabilistic prototype tree PPT (middle). Possible extracted program P_{ROG}, at the time of creation of instruction $I_{1,0}$, the dashed part of P_{ROG} did not exist yet (right).

REPEAT:

$$P_j\left(I_j(P_{ROG_b})\right) = P_j\left(I_j(P_{ROG_b})\right) + c^{lr} \cdot lr \cdot \left(1 - P_j\left(I_j(P_{ROG_b})\right)\right) \quad (7)$$

UNTIL $P\left(P_{ROG_b}\right) \geq P_{TARGET}$

where c^{lr} is a constant influencing the number of iterations. The smaller c^{lr} the higher the approximation precision of P_{TARGET} and the number of required iterations. Setting $c^{lr} = 0.1$ turned out to be a good compromise between precision and speed. And then all adapted vectors $\overrightarrow{P_j}$ are renormalized.

Step 4. **Mutation of Prototype Tree.** All probabilities $P_j(I)$ stored in nodes N_j that were accessed to generate program P_{ROG_b} are mutated with a probability P_{M_p}:

$$P_{M_p} = \frac{P_M}{n \cdot \sqrt{|P_{ROG_b}|}} \quad (8)$$

where the user-defined parameter P_M defines the overall mutation probability, n is the number of instructions in instruction set S and $|P_{ROG_b}|$ denotes the number of nodes in program P_{ROG_b}. Selected probability vector components are then mutated as follows:

$$P_j\left(I\right) = P_j\left(I\right) + mr \cdot \left(1 - P_j\left(I\right)\right) \quad (9)$$

where mr is the mutation rate, another user-defined parameter. Also all mutated vectors $\overrightarrow{P_j}$ are renormalized.

Step 5. **Prototype Tree Pruning.** At the end of each generation the prototype tree is pruned, as described in Section III.B.

Step 6. **Termination Criteria.** Repeat above procedure until a fixed number of program evaluations is reached or a satisfactory solution is found.

Elitist Learning

Elitist learning focuses search on previously discovered promising parts of the search space. The PPT is adapted towards the elitist program ${P_{ROG}}^{el}$. This is realized by replacing the P_{ROG_b} with ${P_{ROG}}^{el}$ in learning from population in Step 3. It is particularly useful with small population sizes and works efficiently in the case of noise-free problems.

D. Parameter optimization by PSO

The Particle Swarm Optimization (PSO) conducts searches using a population of particles which correspond to individuals in evolutionary algorithm (EA). A population of particles is randomly generated initially. Each particle represents a potential solution and has a position represented by a position vector $\mathbf{x_i}$. A swarm of particles moves through the problem space, with the moving velocity of each particle represented by a velocity vector $\mathbf{v_i}$. At each time step, a function f_i representing a quality measure is calculated by using $\mathbf{x_i}$ as input. Each particle keeps track of its own best position, which is associated with the best fitness it has achieved so far in a vector $\mathbf{p_i}$. Furthermore, the best position among all the particles obtained so far in the population is kept track of as $\mathbf{p_g}$. In addition to this global version, another version of PSO keeps track of the best position among all the topological neighbors of a particle. At each time step t, by using the individual best position, $\mathbf{p_i}$, and the global best position, $\mathbf{p_g}(\mathbf{t})$, a new velocity for particle i is updated by

$$\mathbf{v_i}(\mathbf{t}+1) = \mathbf{v_i}(\mathbf{t}) + c_1\phi_1(\mathbf{p_i}(\mathbf{t}) - \mathbf{x_i}(\mathbf{t})) + c_2\phi_2(\mathbf{p_g}(\mathbf{t}) - \mathbf{x_i}(\mathbf{t})) \quad (10)$$

where c_1 and c_2 are positive constant and ϕ_1 and ϕ_2 are uniformly distributed random number in [0,1]. The term $\mathbf{v_i}$ is limited to the range of $\pm\mathbf{v_{max}}$. If the velocity violates this limit, it is set to its proper limit. Changing velocity this way enables the particle i to search around its individual best position, $\mathbf{p_i}$, and global best position, $\mathbf{p_g}$. Based on the updated velocities, each particle changes its position according to the following equation:

$$\mathbf{x_i}(\mathbf{t}+\mathbf{1}) = \mathbf{x_i}(\mathbf{t}) + \mathbf{v_i}(\mathbf{t}+\mathbf{1}). \quad (11)$$

E. The proposed algorithm for designing of hierarchical TS-FS model

Combining the self-organizing and structure learning characteristics of PIPE and the parameter optimization ability of PSO, we propose the following hybrid algorithm for designing the hierarchical TS-FS model.

1) Set the initial values of parameters used in the PIPE and PSO algorithms. 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 parameters used in hierarchical TS-FS model.

2) Do structure optimization using PIPE algorithm as described in subsection III.C, in which the fitness function is calculated by Root Mean Square Error (RMSE).

3) Determine the fuzzy membership function parameters of the non-leaf nodes for each sub-model by using k-means cluster technique;

4) Parameter optimization using PSO as described in subsection III.D. In this step, the tree structure or architecture of hierarchical TS-FS model is fixed, and it is the best tree taken from the end of run of PIPE search. The non-leaf node's fuzzy membership function parameters and the linear weights in the consequent parts of the rules will be optimized by PSO in order to decrease the fitness value of best program;

5) If the maximum number of PSO search is reached, or no better parameter vector is found for a significantly long time (100 steps) then go to step 6); otherwise go to step 4).

6) If satisfactory solution is found, then stop; otherwise go to step 2).

IV. Simulation Studies

The proposed approach has been evaluated for Mackey-Glass chaotic time-series prediction problem. The used parameters in PIPE are shown in Table 1. For all the simulations, the minimum and maximum number of hierarchical layers are predefined as 2 and 4, and each middle variable of the non-leaf node is partitioned into 2 fuzzy sets. The used fuzzy membership function is shown in Eqn. (2). All the free parameters including fuzzy sets membership function parameters and the free parameters in the consequent parts of fuzzy rules are randomly generated at [0,1] initially. It should be noted that the pre-selection of the non-leaf's instruction is experimental. Selecting more instructions will increase the structure/parameter search space and results in a bigger hierarchical TS fuzzy system. For an identification or classification problem, if the input number is n, selecting the maximum instruction $+_N$ as $N = n/3$ is enough according to our experiments. This experimental rule should reduce the search space significantly. The Mackey-Glass chaotic differential delay equation is recognized as a benchmark problem that has been used and reported by a number of researchers for comparing the learning and generalization ability of different models. The Mackey-Glass chaotic time series is generated using the following differential equation:

$$\frac{dx(t)}{dt} = \frac{ax(t-\tau)}{1+x^{10}(t-\tau)} - bx(t) \tag{12}$$

where $a = 0.2$ and $b = 0.1$, $\tau > 17$ the equation shows chaotic behavior. In our simulations, $\tau = 30$ has been adopted.

TABLE I

Parameters Used in the PIPE Algorithm

Parameters	Values
population size PS	100
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

TABLE II

Comparison of the incremental type multilevel FRS (IFRS) [14], the aggregated type mutilevel FRS (AFRS) [14], and the hierarchical TS-FS with randomly initialized free parameters and fuzzy rules [23] for Mackey-Glass time-series prediction

Model	Stage	Rules	Parameters	RMSE training	RMSE testing
IFRS [14]	4	25	58	0.0240	0.0253
AFRS [14]	5	36	78	0.0267	0.0256
H-TS-FS1 [23]	3	28	148	0.0120	0.0129
H-TS-FS2 [23]	2	12	46	0.0145	0.0151
This paper	2	10	36	0.0105	0.0112

To compare with previous works [14], we predicted the value of $x(t+6)$ using the input variables $x(t-30)$, $x(t-24)$, $x(t-18)$, $x(t-12)$, $x(t-6)$ and $x(t)$, where $t = 130$ to $t = 1129$. It corresponds to a 6-input to 1-output mapping.

1000 sample points were used in our study. The first 500 data pairs were used as training data, while the remaining 500 were used to validate the model identified. The used instruction set is $I = \{+_2, +_3, x_0, x_1, x_2, x_3, x_4, x_5\}$, where $x_0, x_1, x_2, x_3, x_4, x_5$ denote $x(t-30)$, $x(t-24)$, $x(t-18)$, $x(t-12)$, $x(t-6)$ and $x(t)$, respectively.

The results are obtained from training the hierarchical TS-FS models using 10 different experiments. The best $RMSE$ value for training and test data sets are 0.0105 and 0.0112, respectively. A comparison has been made to show the actual time-series, the hierarchical TS-FS model output and the prediction error (Fig.3 (top)). Fig.3 (bottom) shows the convergence performance of the best hierarchical TS-FS model. Performance comparison of the different methods for predicting the Mackey-Glass time-series is shown in Table 2.

V. Conclusion

In this paper, an automatic design and optimization method for hierarchical TS-FS is proposed. In the proposed framework, PIPE, PSO and *k*-means clustering algorithms are combined to design an optimal hierarchical TS-FS model. The effectiveness of the proposed methods has been demonstrated through Mackey-Glass time-series forecasting problem. The new method results in a smaller rule-base with good learning ability. Our future research targets on more simulations and real applications for evaluating the proposed method.

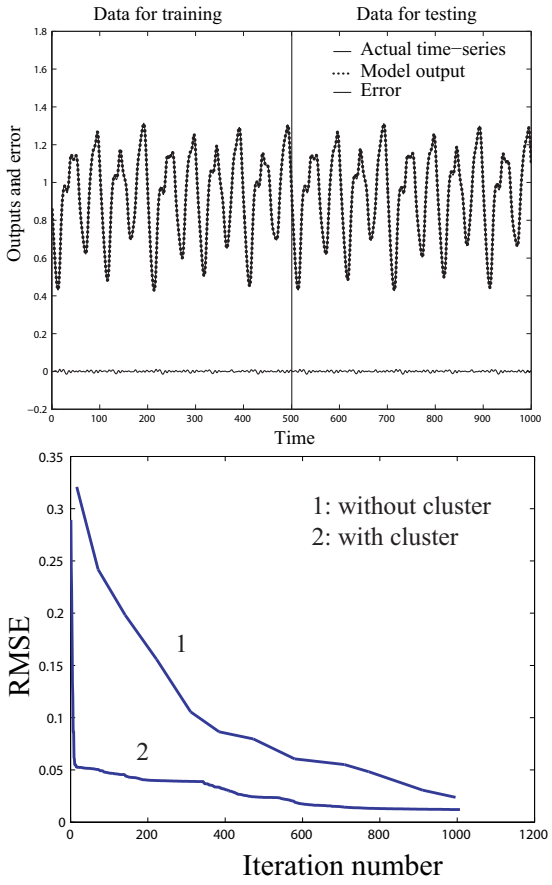

Fig. 3. Actual time-series, model output and prediction error (Top), and the fitness curve for training (bottom).

ACKNOWLEDGMENT

This research was partially supported by the Natural Science Foundation of China under grant number 60573065, and The Provincial Science and Technology Development Program of Shandong under grant number SDSP2004-0720-03.

REFERENCES

[1] X. J. Zeng and M. G. Singh, "Approximation theory of fuzzy system - MIMO case", *IEEE Trans. on Fuzzy Systems*, Vol.3, No.2, pp.219-235, 1995.

[2] X. J. Zeng and M. G. Singh, "Approximation theory of fuzzy systems - SISO case", *IEEE Trans. on Fuzzy Systems*, Vol.2, No.2, pp.162-176, 1994.

[3] A. Abraham, "Adaptation of Fuzzy Inference System Using Neural Learning, Fuzzy System Engineering: Theory and Practice," Nadia Nedjah et al. (Eds.), *Studies in Fuzziness and Soft Computing*, Springer Verlag Germany, ISBN 3-540-25322-X, Chapter 3, pp. 53-83, 2005.

[4] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its application to modeling and control," *IEEE Trans. Syst. Man, Cybern.*, vol. 15, pp. 116-132, 1985.

[5] M. Denna, G. Mauri, A.M. Zanaboni, "Learning fuzzy rules with tabu search-an application to control," *IEEE Trans. on Fuzzy Systems*, vol. 7, no. 2, pp. 295-318, 1999.

[6] G.V. Raju and J. Zhou, "Adaptive hierarchical fuzzy controller," *IEEE Trans. on System, Man and Cybernetics* vol. 23, no. 4, pp. 973-980, 1993.

[7] L.X. Wang, "Analysis and design of hierarchical fuzzy systems," *IEEE Trans. on Fuzzy Systems*, vol. 7, no. 5, pp. 617-624, 1999.

[8] L.X. Wang, "Universal approximation by hierarchical fuzzy systems," *Fuzzy Sets and Systems*, vol. 93, pp. 223-230, 1998.

[9] O. Huwendiek, W. Brockmann, "Function approximation with decomposed fuzzy systems," *Fuzzy Sets and Systems*, vol. 101, pp. 273-286, 1999.

[10] K. Hiroaki et al., "Functional completeness of hierarchical fuzzy modeling," *Information Science*, vol. 110, no. 1-2, pp. 51-60, 1998.

[11] H. Rainer, "Rule generation for hierarchical fuzzy systems," *Proc. of the annual conf. of the North America Fuzzy Information Processing*, pp. 444-449, 1997.

[12] K. Sun-Yuan et al., "Synergistic modeling and applications of hierarchical fuzzy neural networks," *Proceedings of IEEE*, vol. 87, no. 9, pp. 1550-1574, 1999.

[13] M. Brown, K.M., Bossley, D.J. Mills and C.J. Harris, "High dimensional neurofuzzy systems: overcoming the curse of dimensionality," *Proc. 4th Int. Conf. on Fuzzy Systems*, pp. 2139-2146, 1995.

[14] J.-C. Duan, F.-L. Chung, "Multilevel fuzzy relational systems: structure and identification," *Soft Computing*, vol. 6, pp. 71-86, 2002.

[15] M.G. Joo, J.S. Lee, "A class of hierarchical fuzzy systems with constraints on the fuzzy rules," *IEEE Trans. on Fuzzy Systems*, vol. 13, no. 2, pp. 194-203, 2005.

[16] S. Paulo, "Clustering and hierarchization of fuzzy systems", *Soft Computing Journal*, vol. 9, no. 10, pp. 715-731, 2005.

[17] V. Torra, "A review of the construction of hierarchical fuzzy systems," *International Journal of Intelligent Systems*, Vol. 17, pp. 531-543, 2002.

[18] Shimojima, K., Fukuda T., Hasegawa Y., "Self-turning fuzzy modeling with adaptive membership function, rules, and hierarchical structure based on genetic algorithm," *Fuzzy Sets and Systems*, vol. 71, pp. 295-309, 1995.

[19] C. Wei and Li-Xin Wang, "A note on universal approximation by hierarchical fuzzy systems," *Information Science*, vol. 123, pp. 241-248, 2000.

[20] X.-J. Zeng, J.A. Keane, "Approximation Capabilities of Hierarchical Fuzzy Systems," *IEEE Trans. on Fuzzy Systems*, vol. 13, no. 5, pp. 659-672, 2005.

[21] L. C. Lin and G.-Y. Lee, "Hierarchical fuzzy control for C-axis of CNC tuning centers using genetic algorithms," *Journal of Intelligent and Robotic Systems*, vol. 25, no. 3, pp. 255-275, 1999.

[22] Y. Chen, B. Yang and J. Dong, "Automatic design of hierarchical TS-FS models using ant programming and PSO algorithm," *The Eleventh International Conference on Artificial Intelligence: Methodology, Systems, Applications*, LNCS 3192, pp. 285-294, 2004.

[23] Y. Chen, B. Yang, A. Abraham and L. Peng, "Evolutionary Design of Hierarchical TS Fuzzy Models using Evolutionary Algorithms," *IEEE Trans. on Fuzzy Systems*, 2006. (In press)

[24] R. Babuska, Fuzzy modeling and identification, Ph.D. Thesis, University of Delft, the Netherlands, 1996.

[25] Angelov P., D. Filev, "An approach to on-line identification of Takagi-Sugeno fuzzy models," *IEEE Transactions on System, Man, and Cybernetics, part B - Cybernetics*, vol.34, no. 1, pp.484-498, 2004.

[26] Kasabov, N., Song, Q., "DENFIS: Dynamic, evolving neural-fuzzy inference system and its application for time-series prediction," *IEEE Trans. on Fuzzy Systems*, vol. 10, pp. 144-154, 2002.

[27] R. P. Salustowicz and J. Schmidhuber, "Probabilistic incremental program evolution," *Evolutionary Computation*, vol. 2, no. 5, pp. 123-141, 1997.

[28] R. P. Salustowicz and J. Schmidhuber, "Evolving structured programs with hierarchical instructions and skip nodes," In Shavlik, J., editor, *Machine Learning: Proceedings of the Fifteenth International Conference (ICML98)*, pp. 488-496. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1998.

[29] J. Schmidhuber, "On learning how to learn learning strategies," *Technical Report FKI-198-94*, Fakultaet fuer Informatik, Technische Universitaet, Muenchen. Revised January 1995.

[30] M. A. Wiering and J. Schmidhuber, "Solving POMDPs with Levin search and EIRA," *In Saitta, L., editor, Machine Learning: Proceedings of the Thirteenth International Conference, Morgan Kaufmann Publishers, San Francisco, CA*, pp. 534-542, 1996b.

[31] N. L. Cramer, "A representation for the adaptive generation of simple sequential programs," *In Grefenstette, J., editor, Proceedings of an International Conference on Genetic Algorithms and Their Applications, Hillsdale NJ. Lawrence Erlbaum Associates*, pp. 183-187, 1985.