ELSEVIER

Journal of Systems Architecture 47 (2001) 587–599

JOURNAL OF SYSTEMS ARCHITECTURE

www.elsevier.com/locate/sysarc

System identification using evolutionary Markov chain Monte Carlo

Byoung-Tak Zhang *, Dong-Yeon Cho

Artificial Intelligence Lab (SCAI), School of Computer Science and Engineering, Seoul National University, Seoul 151-742, South Korea

Abstract

System identification involves determination of the functional structure of a target system that underlies the observed data. In this paper, we present a probabilistic evolutionary method that optimizes system architectures for the identification of unknown target systems. The method is distinguished from existing evolutionary algorithms (EAs) in that the individuals are generated from a probability distribution as in Markov chain Monte Carlo (MCMC). It is also distinguished from conventional MCMC methods in that the search is population-based as in standard evolutionary algorithms. The effectiveness of this hybrid of evolutionary computation and MCMC is tested on a practical problem, i.e., evolving neural net architectures for the identification of nonlinear dynamic systems. Experimental evidence supports that evolutionary MCMC (or eMCMC) exploits the efficiency of simple evolutionary algorithms while maintaining the robustness of MCMC methods and outperforms either approach used alone.

Keywords: System identification; Markov chain; Monte Carlo

1. Introduction

System identification typically involves constructing function f_A of a target system f given a set D of input–output pairs $(\mathbf{x}_c, f(\mathbf{x}_c))$, i.e., $D = \{(\mathbf{x}_c, f(\mathbf{x}_c)) \,|\, c = 1, \ldots, N\}$. Here $f(\mathbf{x}_c)$ is the observed output of the system given the input $\mathbf{x}_c$. The objective is to find the architecture A^*, i.e., the functional structure and associated parameters, whose output $f_{A^*}(\mathbf{x})$ best predicts the output $f(\mathbf{x})$ of the target system given an arbitrary input $\mathbf{x}$.

Evolutionary computation has been used from its inception for automatic identification of a given system or process. For example, Fogel et al. [6] used simulated evolution to induce programs that predict a sequence of symbols from an environment. Here the target system f is an ideal algorithm that perfectly predicts the next symbol from a sequence of observed symbols, and a model f_A of the system is represented as a finite state machine (FSM). A set D of training cases are given as a sequence of symbols $x_1, x_2, \ldots, x_c, x_{c+1}, \ldots, x_N$. A population of FSMs is exposed to the training cases. For each input symbol x_c, the actual output $f_A(x_c)$ of an FSM is compared with the true output, i.e., the next input symbol $x_{c+1} = f(x_c)$. The fitness of the FSM is measured as, for example,

* Corresponding author. Tel.:+82-2-880-1833; fax:+82-2-883-3595.

E-mail addresses: btzhang@cse.snu.ac.kr, btzhang@bi.snu.ac.kr (B.-T. Zhang), dycho@bi.snu.ac.kr (D.-Y. Cho).

1383-7621/01/$ - see front matter
PII: S1383-7621(01)00017-0

the total number of the differences between the predicted symbols and the true next symbols. Offspring machines are created by randomly mutating each parent machine. Mutations are performed by changing a state transition, adding a state, deleting a state, or the like. The offspring are then evaluated on the fitness cases D and the most fit machines are selected to become the next generation.

A variety of structures have been used as models of target systems. These include sequences of transfer functions [10], graph structures [4], Lisp-like symbolic programs [11], recurrent networks [2], and many others [13]. The application areas range from scientific modeling and engineering design to artificial intelligence and artificial life [5]. Despite this diversity, most existing methods for system identification have been focused on operators for generating new structures. Relatively little effort has been put into algorithmic improvement.

In this paper, we present a new type of evolutionary computation called evolutionary Markov chain Monte Carlo (eMCMC) method. This is a combination of an evolutionary algorithm (EA) and an MCMC method. In contrast to conventional evolutionary algorithms [3], the eMCMC algorithms generate new structures by sampling from a probability distribution, which is similar to MCMC. In contrast to simple MCMC methods [7], the search is based on a population of search points, which is similar to EAs. One of the main advantages of the eMCMC approach is that it exploits the efficiency of EAs while maintaining the theoretical robustness of MCMC methods. The method is illustrated by providing explicit formulae for the distributions of the architecture space in the context of neural systems. In this application, we have two levels of learning. The problem being solved is itself a (neural tree) learning problem. Another level of learning is by the algorithm to learn the probability distribution. This second learning is equivalent to Bayesian learning but done by MCMC.

The effectiveness and robustness of the method are demonstrated on the identification of the dynamical systems. We also study the effect of the population size and probabilistic sampling of offspring in eMCMC, as compared to conventional MCMC and standard evolutionary algorithms. Experimental results show that simple MCMC methods can be improved by adopting the concept of population. This seems due to the increased diversity of individuals and the synergy effect from their combinations. Our results also show that simple evolutionary algorithms can be made more robust by using probabilistic models for offspring generation.

The paper is organized as follows. In Section 2 we describe the architecture of neural systems we evolve for system identification. Section 3 presents the evolutionary MCMC algorithm for evolving the neural systems. Section 4 reports experimental results on the identification of a laser system from real-life data and analyzes the effects of the population sizes and probability distributions on the accuracy and complexity of architectures. Section 5 summarizes our findings from this study.

2. Probabilistic formulation

In this section we describe the class of systems we consider for the identification of unknown systems. This constitutes the search space of the evolutionary algorithms. We then define a probability distribution on the space of system architectures we search through. The evolutionary search procedure itself will be described in Section 3.

2.1. Model architectures

The problem is to identify the functional structure and associated parameters of an unknown target system. This involves constructing functions f_A of a target system f given a set D of input–output pairs $(\mathbf{x}_c, f(\mathbf{x}_c))$, i.e., $D = \{(\mathbf{x}_c, f(\mathbf{x}_c)) \,|\, c = 1, \ldots, N\}$. Here $f(\mathbf{x}_c)$ is the observed output of the target system given the input $\mathbf{x}_c$. If we denote A as an individual system architecture and $\mathscr{A}$ as the set of all possible architectures, then the problem can be formulated as finding the architecture A^* that minimizes some specified objective function:

$$A^* = \min_{A \in \mathscr{A}} F(A), \tag{1}$$

where $F(A)$ is the fitness measure of architecture A. Usually, one takes the sum of the squared errors on the given training data as the fitness measure of the architecture:

$$F(A) = \sum_{c=1}^{N} \|f_A(\mathbf{x}_c) - f(\mathbf{x}_c)\|^2, \tag{2}$$

where $f_A(\mathbf{x}_c)$ is the actual output of the system A for input $\mathbf{x}_c$ and $f(\mathbf{x}_c)$ is the target output. N denotes the size of data set available.

The class of architectures we consider is neural trees. A neural tree is composed of terminal nodes, nonterminal nodes, and weights of connection links between two nodes [27]. An instance of the neural tree is shown in Fig. 1. The nonterminal nodes represent neural units and the neuron type is an element of the basis function set $\mathscr{F} = \{\text{neuron types}\}$. Each terminal node is labeled with an element from the terminal set $\mathscr{T} = \{x_0, x_1, x_2, \ldots, x_n\}$, where x_i is the ith component of the external input $\mathbf{x}$ and x_0 is fixed to $+1$. All nonterminal nodes have just one fixed input, $+1$, as the externally applied bias. In the neural tree shown in Fig. 1, the function set and terminal set are given as $\mathscr{F} = \{\Sigma, \Pi\}$, $\mathscr{T} = \{x_0, x_1, x_2, x_3, x_4\}$, where the meaning of sigma (Σ) and pi (Π) units are defined below.

Each link (j, i) represents a directed connection from node j to node i, where node i is parent of node j and node j is child of node i. There is also a value w_{ij} which is associated with each link. In this neural tree, the root node is an output unit and the terminal nodes are input units. The depth of a neural tree is defined as the longest path length from the root node to any terminal node of the tree.

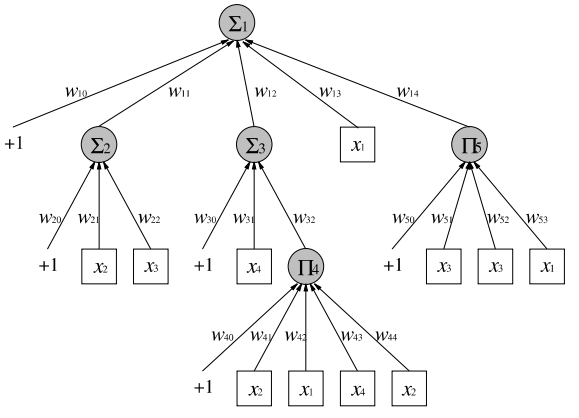


Fig. 1. The structure of a neural tree whose depth is 3. The tree consists of one output unit (root), four hidden units (two different neuron types), and four input units distributed over terminal nodes.

Each nonterminal node gets input signals from maximum $b_{\max} + 1$ child nodes including a bias and produces a single output. Different neuron types are distinguished in the way that the net inputs are computed. One of the most popular neuron types is the sigma unit, which computes the sum of weighted inputs from the lower layer by

$$\text{net}_i = \sum_j w_{ij} y_j, \tag{3}$$

where y_j are the inputs to the ith neuron. Another useful neuron type is the pi unit, which calculates the product of weighted inputs from the lower layer as

$$\text{net}_i = \prod_j w_{ij} y_j, \tag{4}$$

where y_j are the inputs to i. The output of a neuron is computed by the sigmoid transfer function

$$y_i = f(\text{net}_i) = \frac{1}{1 + \mathrm{e}^{-\text{net}_i}}, \tag{5}$$

where net_i is the net input to the unit computed by Eq. (3) or (4).

Neural trees can represent a broad class of feedforward networks that have irregular connectivity and not-strictly layered structures [27]. The tree structure allows for easy exchange of substructures by standard subtree variation operators without destroying building blocks. This is contrasted to other evolutionary methods for neural network design [15,24–26] and conventional statistical methods for model selection [1,8,21] which are based on flat structures of models.

2.2. Defining the probability distributions

To find the fittest architecture by evolutionary search, we first define the probability distributions of neural trees for data. Given the training data $D = \{(\mathbf{x}_c, y_c)\}_{c=1}^{N}$, a neural architecture A is

assumed to represent the following input–output mapping:

$$y_c = f_A(\mathbf{x}_c) + \epsilon. \tag{6}$$

Here, the noise ϵ is assumed to be zero-mean Gaussian with standard deviation σ. Note also that the architecture A of a neural tree is specified by the depth d of the tree, the number k of nodes, and the connection weights $\mathbf{w}$, i.e., $A = (\mathbf{w}, k, d)$. Here, we assume that any trees that have identical depth, number of nodes and weights have the same prior probability.

If we assume that data items are independent of each other, then the likelihood of the neural tree can be expressed as follows:

$$\begin{aligned} P(D|\mathbf{w},k,d) &= \prod_{c=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_c - f_{(\mathbf{w},k,d)}(\mathbf{x}_c))^2}{2\sigma^2} \right) \\ &= \left(\frac{1}{\sqrt{2\pi}\sigma} \right)^N \exp\left(-\frac{\sum_{c=1}^{N}(y_c - f_{(\mathbf{w},k,d)}(\mathbf{x}_c))^2}{2\sigma^2} \right), \end{aligned} \tag{7}$$

where $f_{(\mathbf{w},k,d)}(\mathbf{x}_c)$ is the actual output of the neural tree with architecture $A = (\mathbf{w}, k, d)$, and y_c is the target output for input $\mathbf{x}_c$. From the system identification point of view, identification of the target system involves the identification of $A = (\mathbf{w}, k, d)$ values starting from their prior values. We define the following prior probability for weights of the neural tree:

$$\begin{aligned} P(\mathbf{w}|k,d) &= \prod_{j=1}^{k-1} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{w_j^2}{2} \right) \\ &= \left(\frac{1}{\sqrt{2\pi}} \right)^{k-1} \exp\left(-\frac{\sum_{j=1}^{k-1} w_j^2}{2} \right), \end{aligned} \tag{8}$$

where the components of the weight vector are assumed to be independent of each other and distributed according to zero-mean Gaussian with standard deviation 1. This prior says that smaller weights are preferred as in the weight decay procedure [12]. We assume that the number of nodes in the neural tree is uniformly distributed over the interval $[k_{\min}, k_{\max}]$.

$$P(k|d) = \frac{1}{k_{\max} - k_{\min} + 1}, \tag{9}$$

where $k_{\min} = 2d + 1$ and $k_{\max} = (b_{\max}^{d+1} + b_{\max}^{d} - 2)/(b_{\max} - 1)$. This implies that all trees whose depth is d have the same prior probability for the number of nodes. However, the trees which consist of more nodes tend to have lower prior values for the weight vector (8). We also assume that the depth of the neural tree is distributed according to the following truncated Poisson distribution:

$$P(d) = \begin{cases} \frac{(\lambda - d_{\min})^{(d-d_{\min})} \exp(-\lambda + d_{\min})}{(d-d_{\min})!} \\ \quad \text{if } d_{\min} \leqslant d \leqslant d_{\max} - 1, \\ 1 - \sum_{j=d_{\min}}^{d_{\max}-1} P(j) \\ \quad \text{if } d = d_{\max}, \\ 0 \\ \quad \text{otherwise.} \end{cases} \tag{10}$$

With $d_{\min}$, $d_{\max}$, and λ, we can eliminate both too simple and too complex trees from our consideration to generate trees with moderate depth. Substituting Eqs. (7)–(10) into Eq. (15), we obtain the following posterior probability (up to a normalizing constant) for the neural tree:

$$\begin{aligned} P(A|D) &\propto P(D|\mathbf{w},k,d)P(\mathbf{w}|k,d)P(k|d)P(d) \\ &= \left(\frac{1}{\sqrt{2\pi}\sigma} \right)^N \exp\left(-\frac{\sum_{c=1}^{N}(y_c - f_{(\mathbf{w},k,d)}(\mathbf{x}_c))^2}{2\sigma^2} \right) \\ &\quad \times \left(\frac{1}{\sqrt{2\pi}} \right)^{k-1} \exp\left(-\frac{\sum_{j=1}^{k-1} w_j^2}{2} \right) \frac{1}{k_{\max} - k_{\min} + 1} \\ &\quad \times \frac{(\lambda - d_{\min})^{(d-d_{\min})} \exp(-\lambda + d_{\min})}{(d - d_{\min})!}, \end{aligned} \tag{11}$$

where the parameter σ should be chosen to balance the likelihood with priors. A more sophisticated method can be used to change σ dynamically according to the sample distribution, but we fixed it in this work. Given this probability distribution, we show in Section 3 that solving the minimization problem (1) is equivalent to maximizing (11).

3. The evolutionary MCMC algorithm

The optimization problem is formally stated and an evolutionary algorithm for solving the problem is presented. We also discuss the connections between the algorithm and other conventional methods.

3.1. Architecture optimization as Bayesian inference

Bayes theorem provides a direct method for computing the posterior probability $P(A|D)$ of each architecture A given the observed training data D. By Bayes theorem [20], we have

$$P(A|D) = \frac{P(D|A)P(A)}{P(D)}, \tag{12}$$

where $P(A)$ is the prior probability for the architecture, $P(D|A)$ is the likelihood of the model for the data, and $P(D)$ is a normalizing constant computed as

$$P(D) = \int P(D|A)P(A)\,\mathrm{d}A. \tag{13}$$

Since we are interested in finding the architecture that maximizes the posterior probability,

$$A^* = \arg\max_A P(A|D), \tag{14}$$

the posterior probability (12) of a neural tree can be written as

$$\begin{aligned} P(A|D) &\propto P(D|A)P(A) \\ &= P(D|\mathbf{w},k,d)P(\mathbf{w},k,d) \\ &= P(D|\mathbf{w},k,d)P(\mathbf{w}|k,d)P(k|d)P(d). \end{aligned} \tag{15}$$

In the equation above, we have used the fact that the architecture A of a neural tree is specified by the depth d, the number k of nodes and the connection weights $\mathbf{w}$, i.e., $A = (\mathbf{w}, k, d)$.

Given this, the system identification problem is formulated as an optimization problem that finds the maximum a posteriori probability (MAP):

$$\begin{aligned} A^* &= \arg\max_A P(A|D) \\ &= \arg\max_{\mathbf{w},k,d} P(D|\mathbf{w},k,d)P(\mathbf{w}|k,d)P(k|d)P(d), \end{aligned} \tag{16}$$

where the probability distributions $P(D|\mathbf{w},k,d)$, $P(\mathbf{w}|k,d)$, $P(k|d)$, and $P(d)$ are defined as (7), (8), (9), and (10), respectively.

One simple method for solving this problem is to generate samples A from the distribution $P(A|D)$ and select the best. This straightforward but powerful method builds the basis of all simulation-based optimization methods such as Monte Carlo optimization [23]. If we can generate independent samples from the target distribution, laws of large numbers ensure that the approximation can be made as accurate as desired by increasing the sample size [7]. In general, drawing samples independently from the target distribution is not feasible, since the distribution can be quite non-standard. However the sample need not necessarily be independent. The samples can be generated by any process which draws samples throughout the support of the target distribution in the correct proportions. One way of doing this is through a Markov chain having $P(A|D)$ as its stationary distribution. This is then MCMC.

Traditional MCMC algorithms generate a single chain of samples. This might not be the best strategy. We extend the basic MCMC by introducing the notion of population as used in evolutionary computation. This leads to an evolutionary MCMC algorithm, as described in Section 3.2.

3.2. The eMCMC algorithm

The eMCMC algorithm is summarized in Fig. 2. In essence, it repeatedly generates a population of individuals as in standard evolutionary

1. Set the training set $D = \{(\mathbf{x}_c, y_c) \mid c = 1, ..., N\}$. Generate $\mathcal{A}^0 = \{A_1^0, ..., A_M^0\}$ from prior distribution $P_0(A)$. Set generation count $t \leftarrow 0$.
2. Estimate posterior distribution $P_t(A|D)$ of the individuals $A_i^t = (\mathbf{w}_i^t, k_i^t, d_i^t)$ in $\mathcal{A}^t$. Set the best individual A_{best}^t.
3. Generate L variations $\mathcal{A}' = \{A_1', ..., A_L'\}$ by sampling from $P_t(A|D)$ and training the weights w of the individuals A.
4. Select M best individuals from $\mathcal{A}'$ into $\mathcal{A}^{t+1} = \{A_1^{t+1}, ..., A_M^{t+1}\}$ based on $P(A_i'|D)$.
5. If the termination condition is met, then stop. Otherwise, set $t \leftarrow t+1$ and go to Step 2.

Fig. 2. Outline of the eMCMC algorithm for learning neural trees.

algorithms. The difference is that eMCMC attempts to sample from the posterior fitness distribution of individuals in the population. The probability models for the component distributions are as given in Section 3.1.

In specific, we maintain a population $\mathscr{A}(t)$ of individuals A_i (neural trees in this case) at tth generation:

$$\mathscr{A}(t) = \{A_1, A_2, \ldots, A_M\}, \tag{17}$$

where M is the population size. The main difference from other EAs is that we have explicit probability distributions on the individual space and the individuals are generated from this distribution. Thus, the initial population $\mathscr{A}(0)$ is created according to the prior probability $P_0(A)$ of individuals. This is performed by sampling first a value d_i for the depth of a tree from the Poisson distribution (10) and then a value k_i for the number of nodes from the uniform distribution (9) using the values $k_{\min}$ and $k_{\max}$ for the given d_i. Finally, we sample $\mathbf{w}_i$ from the Gaussian distribution (8) using k_i value. In each generation t, the error $E_i(t)$ of neural trees is evaluated as follows:

$$E_i(t) = \sum_{c=1}^{N} (y_c - f_{A_i}(\mathbf{x}_c))^2. \tag{18}$$

This value builds the basis for calculating the likelihood (7) of the neural tree in the population. Finally, the posterior probability of each neural tree is computed by Eq. (11).

To construct the next generation $\mathscr{A}(g+1)$, a candidate tree A_i' is first created from the parent tree A_i in the current population by variation operators such as recombination and mutation. This means that the proposal distribution in the MCMC method is replaced with a distribution $q(A_i'|A_i)$ which is sampled by variation operators. This is justified since under mild conditions the stationary distribution of the Markov chain can remain the same although the proposal distribution has changed [7]. This may affect the efficiency of sampling, but not the convergence property. If we assume symmetricity in the proposal distribution, then the candidate tree is accepted with the following probability:

$$\alpha(A_i, A_i') = \min\left\{1, \frac{P(A_i'|D)}{P(A_i|D)}\right\}, \tag{19}$$

which is called the acceptance probability. This acceptance rule says that the candidate tree is always accepted when the posterior probability of the candidate tree is higher than that of the parent; otherwise, it is accepted according to the ratio of two probabilities. If the candidate tree is accepted, A_i' is copied into the next generation. If candidate is rejected, then A_i is copied into the next generation.

Two major variation operators are applied to the parent trees for generating candidate trees. First, recombination operators swap two subtrees chosen at random from the parent tree A_i and another tree A_j $(i \neq j)$, which are selected randomly from the current population to create the candidate tree A_i'. Second, mutation operators change the type of nonterminal nodes or the index of incoming units in the subtree which is also chosen randomly from the parent tree. Other kinds of mutations can also be defined (see below). The probabilities for applying these operators are p_c and p_m, respectively. These mating steps are performed iteratively until L individuals are produced. It should be noted that the use of variation operators in eMCMC is different from the traditional use of them in conventional evolutionary algorithms. The eMCMC algorithm is based on building a probability distribution and sampling from it, and here we use crossover/mutation-like variation operators as a technique for sampling from the distribution. This is because the direct sampling from the probability distribution is intractable and we instead use a proposal distribution as in MCMC.

Candidate trees can also be created by the insert and delete (variants of mutation) operators. These operators are limited to modifying the structure of neural trees one node at a time and motivated from [1,8]. This is an exploitative search in the sense the candidate trees are sought near the current tree in the search space. The insert operator adds a random terminal node to a randomly chosen nonterminal node (see Fig. 3(a)). If the nonterminal node has $b_{\max}+1$ branches, one terminal node of the children nodes is changed into a nonterminal node and the terminal node becomes

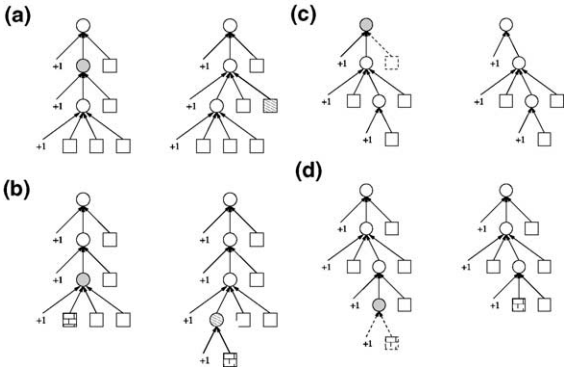


Fig. 3. Insert (a and b) and delete (c and d) operation. These operators are limited to modifying the structure of neural trees one node at a time. In the example trees shown, it is assumed that the maximum number of branches is bounded to $b_{\max} = 3$.

the child node of the new nonterminal node (Fig. 3(b)). The delete operator removes a random terminal node from the child nodes of a randomly selected nonterminal node (Fig. 3(c)). If the nonterminal node has only one child node, the nonterminal node is removed and the child node is linked to the parent node of the nonterminal node (Fig. 3(d)). The probabilities for applying these operators are p_{i} and p_{d}, respectively. The mutation operator is applied to not a random subtree but a random node. For example, pi (Π) node can be changed into sigma (Σ) node, or x_1 can be changed into x_3.

Weights of a neural tree are adjusted through a stochastic hill-climbing. All components of the weight vector $\mathbf{w}$ are changed just once in a random order by Gaussian mutation

$$w'_j = w_j + \mathrm{N}(0,1), \quad j = 1, 2, \ldots, k_i - 1, \tag{20}$$

where k_i is the number of nodes in tree A_i and $\mathrm{N}(0,1)$ is a normal distribution with mean 0 and variance 1. Each change of the weight is also accepted by Eq. (19).

The offspring population $\mathscr{A}'(g)$ is obtained through the above procedure and we finally generate the parent population $\mathscr{A}(g+1)$ of the next generation by selecting the best M individuals which have higher posterior probabilities from $\mathscr{A}'(g)$. This is similar to the (μ, λ) evolution strategy [3].

3.3. Connections to existing methods

The connections of the evolutionary MCMC algorithm to other existing methods for solving the architecture optimization problem should be clear. One of the most interesting features of eMCMC as an evolutionary algorithm is that it is based on probabilistic modeling of the search space. This allows that the offspring are generated according to the probability distribution of the individuals, which is similar to MCMC, rather than simply recombining existing individuals and then using these as candidates for new populations. This property of explicit probabilistic modeling of the search space is related with the recently introduced class of estimation of distribution algorithms or EDAs [16,18].

In eMCMC, a probability model of the search space is pre-specified, their parameters are learned during evolution, and new points are sampled from the regions with higher probability. In this sense, the current implementation of eMCMC is a fixed-model distribution estimation algorithm (EDA), as in FDA of Muehlenbein et al. [17], where the joint probability model used to estimate the true multivariate distribution does not change from generation to generation. Recently, some methods have been presented that adapt dependency models during evolution using graphical models, such as Bayesian networks [19] and Helmholtz machines [31]. A difference of eMCMC from other probabilistic model-building evolutionary algorithms is that we also use crossover/mutation-like variation operators for sampling from the distribution. This is because the direct sampling from the probability distribution is intractable and we instead use a proposal distribution as in MCMC.

It should also be noted that the Bayesian formulation of evolutionary optimization process distinguishes eMCMCs from other EDAs. In the Bayesian formulation, the prior distribution is taken into account as well as the likelihood [28,29]. Since priors can be set uniform if unknown, this provides an additional flexibility that makes evolutionary search more efficient and robust, as demonstrated in Section 4. To see the general benefits of probabilistic modeling in evolutionary algorithms, we also compare the results of eMCMC to those of simple evolutionary algorithms

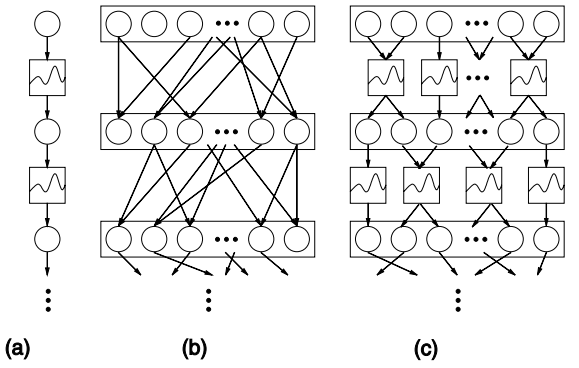


Fig. 4. Comparison of a single-chain MCMC, a conventional EA, and an eMCMC. (a) MCMC generates a single chain using probabilistic sampling. (b) EA generates multiple chains and does not use probability distributions to produce offspring. (c) eMCMC is a hybrid of EA and MCMC that generates multiple chains by sampling offspring from probability distributions.

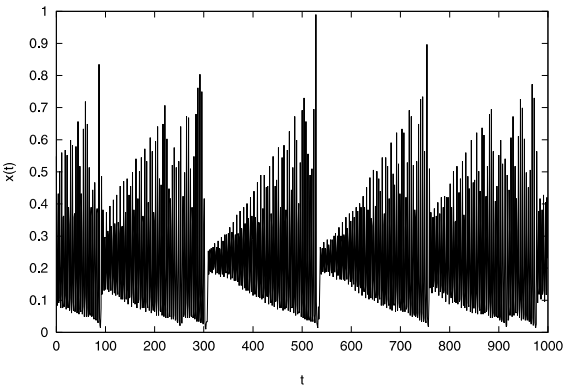


Fig. 5. Time series data generated from the far-infrared NH_3 laser system.

where the probability distributions are not used. Figs. 4(b) and (c) schematically compare the EA and eMCMC methods.

It should also be noted that the optimization problem (16) can also be solved by MCMC methods [22,23]. However, conventional MCMC methods, such as Metropolis algorithms, does not use the notion of population (see Fig. 4(a)). That is, they produce a single chain by generating a new search point from a single existing point. In contrast, the eMCMC algorithm generates multiple chains which are mixed each other as generation goes on [14]. Its simplest form, i.e., the eMCMC with population size 1, can be regarded as a Metropolis algorithm [30]. Thus, we also study the effects of populations by comparing the performances of eMCMC when they are run with different population sizes, including the population size of 1 (which is equivalent to a conventional MCMC method) as a special case.

4. Empirical results and analysis

4.1. The laser system and the yearly sunspot numbers

Experiments have been performed for the identification of a far-infrared NH_3 laser system [9] and a sunspot time series. The laser data were used as a benchmark in the 1992 Santa Fe time series competition. The sunspot data represent the yearly sunspot numbers from 1700 to 1999. [1] The input attributes of the data sets were linearly scaled into the interval $[0.01, 0.99]$, as shown in Figs. 5 and 6.

For the laser data, we used the first 500 data points to evolve the neural tree architectures and the rest 500 data points to test the predictive accuracy. The maximum number of branches of a nonterminal node was $b_{\max} = 3$, which was the same as the input size. The standard deviation of the noise was $\sigma = 0.05$. We set the minimum depth of trees $d_{\min} = 3$, the maximum depth of trees $d_{\max} = 7$, and the average depth of trees $\lambda = 5$. The maximum number of evaluations was $E_{\max} = 10^6$. The parent population size was $M = 50$ and the candidate population size was $L = 100$. Crossover probability was $p_c = 2/3$ and subtree mutation probability was $p_m = 1/3$. For the sunspot data, the period from 1700 to 1920 was used as the training set (221 data) and the period from 1921 to 1999 as the test set (79 data). Experimental setup is same as that of the laser data except for $b_{\max} = 4$, $d_{\min} = 2$, and $d_{\max} = 6$. The choice of these parameters is based on empirical observation.

The performance of eMCMC was compared with those of two other algorithms. One is the

[1] These data are available from the Sunspot Index Data Center (SIDC), http://sidc.oma.be/.

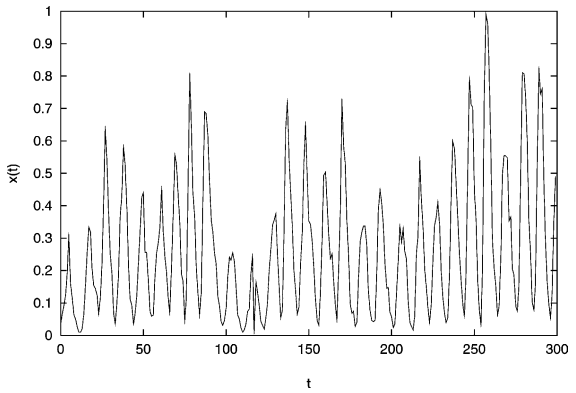


Fig. 6. The yearly sunspot data from 1700 to 1999.

MCMC where only a single individual is evolved. The probability distributions were used as in eMCMC, but no crossover is applied since there is only one individual being evolved ($M = L = 1$). Instead of crossover and subtree mutation, the insert, delete, and node mutation operators were used to propose new individuals. The probability of each operator is 1/3, i.e., $p_i = p_d = p_m = 1/3$. By comparing eMCMC and MCMC we can study the effect of population size in eMCMC. The second algorithm compared is the simple evolutionary algorithm in which no probabilistic model is used but multiple individuals are evolved. Here, the initial population is created randomly, i.e., the initial depth of a tree is sampled from a uniform distribution between $d_{\min}$ and $d_{\max}$ and the initial number of nodes is uniformly sampled between $k_{\min}$ and $k_{\max}$ for the given depth. The weights are also initialized with random values selected uniformly from the range $[-1, 1]$. The error of neural trees (18) is used as a fitness measure. The same crossover and mutation operators as employed in eMCMC are used to create new individuals which are accepted with acceptance probability 1. Weight adjustment values are sampled randomly from $[-1, 1]$ and each change of the weight is accepted if the error is decreased. Other settings are identical with eMCMC. In the following, this algorithm will be denoted simply as EA.

4.2. Results

Fig. 7 shows the results for the algorithms in comparison, i.e. MCMCs (eMCMC with population size one), eMCMCs ($M \geqslant 2$), and EAs (for $M \geqslant 2$). All the methods were given the same computational resource in the total number of function evaluations. Shown are the normalized MSE (NMSE) values of the best neural trees for the training and test data set.

We investigated the effect of population size (i.e., bigger than 1) on the performance of probabilistic evolutionary algorithms. The results are summarized in Fig. 7. It can be seen that all the NMSE values of eMCMCs for both data sets are smaller than that of MCMC. Recalling that the same computing resource was given to all the

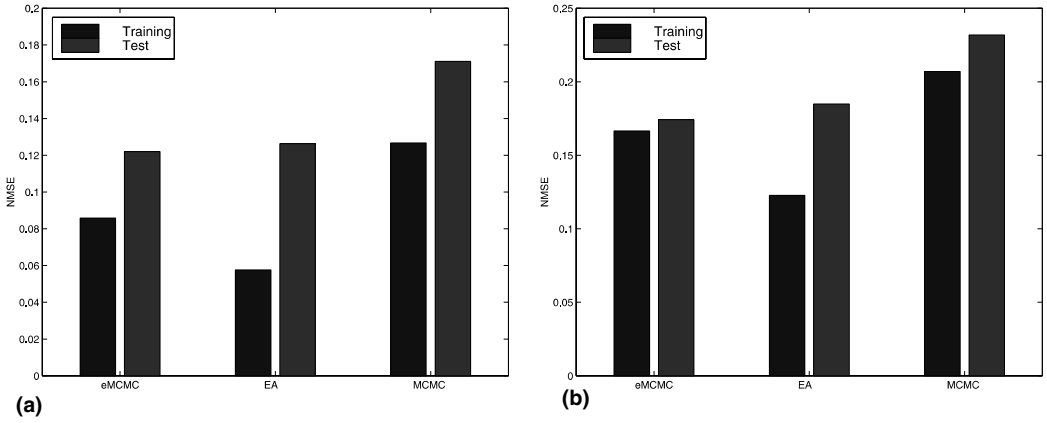


Fig. 7. NMSE values for the training and test set of laser (a) and sunspot (b) data.

variants, this indicates that using multiple individuals and mixing them is a better strategy for optimization than generating a single chain of individuals. The results in Fig. 7 also show that eMCMCs achieve better generalization than EAs although their training performance may worse than that of EAs (as in the case of the laser data). This demonstrates the capability of avoiding overfitting in eMCMC.

The effect of probabilistic evolution was studied further by comparing the fitness and complexity curves of eMCMCs and simple EAs during evolution. The results are shown in Figs. 8 and 9. Shown are the average MSE values for the best individuals at each generation. Fig. 8 compares the MSE values for the nonprobabilistic EAs (conventional EAs) and the probabilistic EAs (eMCMC). The left plot shows the performance on the laser data, and the right plot on the sunspot data. The general trend can be seen that eMCMC is slower than EA in its improvement at early stages, but after some initial generations (equivalent to 200,000 evaluations in this particular case) eMCMC starts to approach the simple EA for the training data and finally has better performance than that of the simple EA for the test data. Fig. 9 compares the evolution of the number of nodes of the architectures and the squared sum of weights evolved by eMCMCs and EAs for both data sets. These results show the effect of priors. The best solutions for eMCMCs in early stage have large structures. This is the reason why the errors of

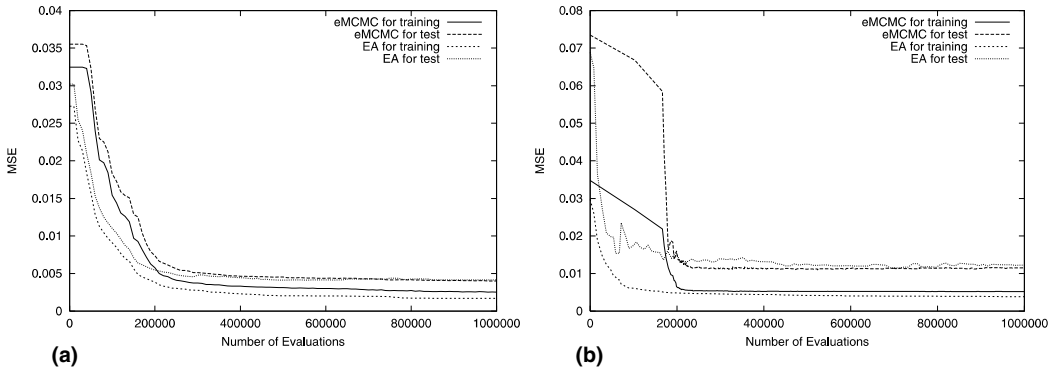


Fig. 8. Comparison of MSE values for the nonprobabilistic EAs (conventional EAs) and the probabilistic EAs (eMCMCs). (Left) Performance on the laser data, (right) performance on the sunspot data.

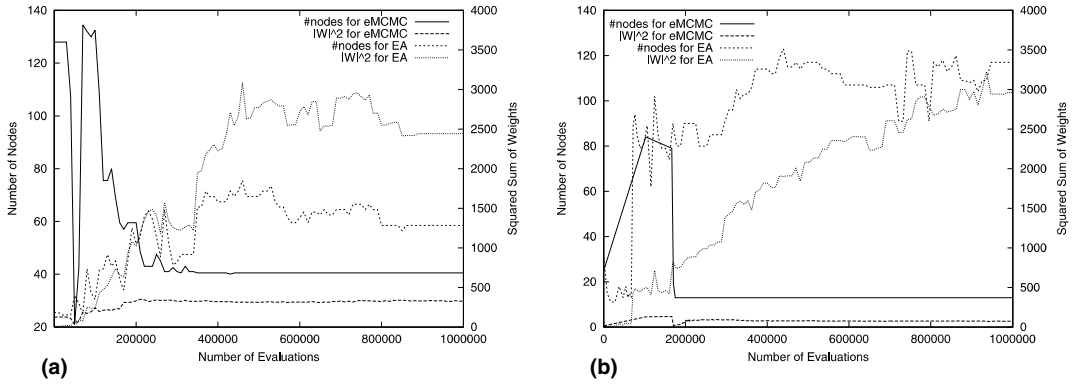


Fig. 9. Comparison of solution complexities for the nonprobabilistic EAs (conventional EAs) and the probabilistic EAs (eMCMC). (a) Laser, (b) sunspot.

eMCMCs are slowly decreased. However, the size of the best trees is reduced rapidly. This seems due to the fact that the prior prefers to simple structures with small weights. In effect, eMCMCs generated trees having approximately 40 nodes for the laser data and 20 nodes for the sunspot data. Moreover, the squared sum of weight values for eMCMCs stay almost fixed as generation goes on. In contrast, the number of nodes and weight values for EAs grow very fast especially in the sunspot data. This demonstrates the advantages of probabilistic sampling in that it avoids exponential growth of model complexities which may (and generally) occur in simple evolutionary algorithms that use variable-size representations [11,27].

In standard evolutionary algorithms, it is usually difficult to incorporate the prior knowledge about the fit individuals explicitly to influence the evolutionary process. However, in eMCMC, it is relatively easy to specify the prior knowledge through the prior probability. This allows for effective local search. Of course, the standard evolutionary algorithms are also able to search locally by using hill-climbing on the weight space and keeping the best individual in the next generation (elitism). However, the straightforward

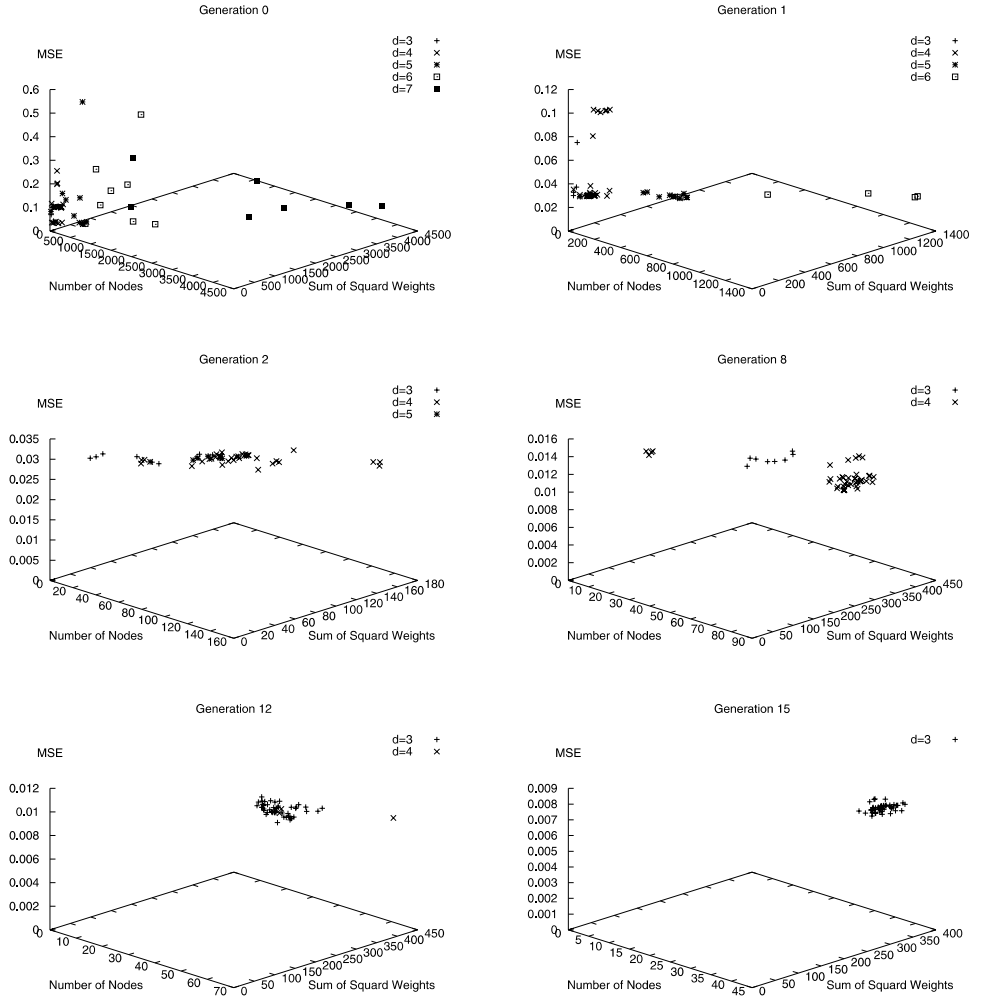


Fig. 10. Distribution of neural trees in the search space in terms of weights, the number of nodes, and the depth of trees with mean squared errors for the laser data.

hill-climbing tends to increase the complexity of individuals as the iteration repeats as we have seen in Fig. 9 and thus, eventually, the tree structures are overfitted to the training data. This ill behavior can be avoided by using the prior probability to control the tree complexity.

4.3. Analysis of convergence properties

A general approach to monitoring convergence in the MCMC method is based on comparing several sequences drawn from different starting points and checking that they are indistinguishable [7]. Fig. 10 shows the location of trees in the search space defined by the weights, the number of nodes, and the depth of trees with their mean squared errors. There are diverse trees in the initial population. However, note that bigger trees disappear rapidly in the early generations. For example, trees whose depth is 7 or 6 vanish in generations 1 and 2, respectively. As the generation goes by, individual trees make some groups (generation 8) and then they are merged little by little (generation 12). Finally, whole trees converge in one particular area and they have the same depth, 3 in this case (generation 15). Note also that the error values of trees decrease gradually. From this generation, local search is performed to find better models but their complexities are not increased as shown in Fig. 9.

5. Conclusions

In this paper, we investigated a probabilistic evolutionary algorithm called eMCMC. It is derived from a Bayesian framework of evolutionary computation and implemented as a hybrid of evolutionary algorithms and MCMC. It was applied to evolving neural trees for the identification of dynamic systems underlying time series data. The performance of eMCMC was analyzed in view of using the population and the prior probability distributions. The experimental results show that using multiple individuals as in eMCMC, contrary to the conventional MCMC methods using single Markov chains, is of benefit in finding better solutions more efficiently. This is attributed to the additional diversity introduced by the population. Compared to conventional evolutionary algorithms, the probabilistic evolutionary algorithms, such as eMCMC, allow the background knowledge about the given data to be incorporated in the evolutionary procedure, and thus local search can be performed more effectively without overfitting to the training data.

Acknowledgements

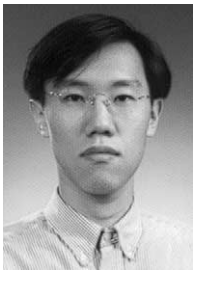
This research was supported in part by the Korea Ministry of Science and Technology through KISTEP under grant BR-2-1-G-06 and by the Korea Ministry of Education under BK21-IT Program.

References

[1] C. Andrieu, N. de Freitas, A. Doucet, Robust full Bayesian methods for neural networks, in: Advances in Neural Information Processing Systems, vol. 12, MIT Press, Cambridge, MA, 2000, pp. 379–385.
[2] P.J. Angeline, G.M. Saunders, J.B. Pollack, An evolutionary algorithm that constructs recurrent neural networks, IEEE Transactions on Neural Networks 5 (1) (1994) 54–65.
[3] T. Bäck, Evolutionary Algorithms in Theory and Practice, Oxford University Press, Oxford, UK, 1996.
[4] D.B. Fogel, System Identification through Simulated Evolution: A Machine Learning Approach to Modeling, Ginn Press, Needham, MA, 1991.
[5] D.B. Fogel (Ed.), Evolutionary Computation: The Fossil Record, IEEE Press, Piscataway, NJ, 1998.
[6] L.B. Fogel, A.J. Owens, M.J. Walsh (Eds.), Artificial Intelligence through Simulated Evolution, Wiley, New York, NY, 1966.
[7] W.R. Gilks, S. Richardson, D.J. Spiegelhalter, Markov Chain Monte Carlo in Practice, Chapman & Hall, London, 1996.
[8] P.J. Green, Reversible jump Markov chain Monte Carlo computation and Bayesian model determination, Biometrika 82 (4) (1995) 711–732.
[9] H. Hübner, C.O. Weiss, N.B. Abraham, D. Tang, Lorenz-like chaos in NH_3-FIR laser, in: Time Series Prediction: Forecasting the Future and Understanding the Past, Addison-Wesley, Reading, MA, 1993, pp. 73–104.
[10] H. Kaufman, An experimental investigation of process identification by competitive evolution, IEEE Transactions on Systems Science and Cybernetics 3 (1) (1967) 11–16.
[11] J.R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, MA, 1992.

[12] A. Krogh, J.A. Hertz, A simple weight decay can improve generalization, in: Advances in Neural Information Processing Systems, vol. 4, MIT Press, Cambridge, MA, 1992, pp. 950–957.
[13] M. Kreutz, A.M. Reimetz, B. Sendhoff, C. Weihs, W. von Seelen, Optimisation of density estimation models with evolutionary algorithms, in: Parallel Problem Solving from Nature, Lecture Notes in Computer Science, vol. 1498, Springer, Berlin, 1998, pp. 998–1007.
[14] S.-E. Lee, B.-T. Zhang, A. Doucet, Convergence properties of Bayesian evolutionary algorithms with population size greater than 1, in: Proceedings of the 2001 Congress on Evolutionary Computation, vol. 1, pp. 326–331.
[15] D. Montana, L. Davis, Training feedforward neural networks using genetic algorithms, in: Proceedings of the International Joint Conference on Artificial Intelligence.
[16] H. Mühlenbein, T. Mahnig, A. Ochoa, Schemata, distributions and graphical models in evolutionary optimization, Journal of Heuristics 5 (1999) 215–247.
[17] H. Mühlenbein, T. Mahnig, FDA – a scalable evolutionary algorithm for the optimization of additively decomposed functions, Evolutionary Computation 7 (4) (1999) 353–376.
[18] M. Pelikan, H. Mühlenbein, The bivariate marginal distribution algorithm, in: Advances in Soft Computing – Engineering Design and Manufacturing, Springer, London, 1999, pp. 521–535.
[19] M. Pelikan, D.E. Goldberg, E. Cantú-Paz, Linkage problem, distribution estimation, and Bayesian networks, Evolutionary Computation 8 (3) (2000) 311–340.
[20] S.J. Press, Bayesian Statistics: Principles, Models, and Applications, Wiley, New York, 1989.
[21] S. Richardson, P.J. Green, On Bayesian analysis of mixtures with an unknown number of components (with discussion), Journal of Royal Statistics Society B 59 (4) (1997) 731–792.
[22] D. Rios Insua, P. Müller, Feedforward neural networks for nonparametric regression, in: Practical Nonparametric and Semiparametric Bayesian Statistics, Lecture Notes in Statistics, vol. 133, Springer, Berlin, 1998, pp. 181–194.
[23] C.P. Robert, G. Casella, Monte Carlo Statistical Methods, Springer, New York, 1999.
[24] J.D. Schaffer, D. Whitley, L.J. Eshelman, Combinations of genetic algorithms and neural networks: a survey of the state of the art, in: Proceedings the International Workshop on Combinations of Genetic Algorithms and Neural Networks, pp. 1–37.
[25] D. Whitley, T. Starkweather, C. Bogart, Genetic algorithms and neural networks: optimizing connections and connectivity, Parallel Computing 14 (1990) 347–361.
[26] X. Yao, Evolutionary artificial neural networks, International Journal of Neural Systems 4 (3) (1993) 203–222.
[27] B.-T. Zhang, P. Ohm, H. Mühlenbein, Evolutionary induction of sparse neural trees, Evolutionary Computation 5 (2) (1997) 213–236.
[28] B.-T. Zhang, A Bayesian framework for evolutionary computation, in: Proceedings of the 1999 Congress on Evolutionary Computation, vol. 1, pp. 722–728.
[29] B.-T. Zhang, Bayesian evolutionary algorithms for learning and optimization, in: Proceedings of the 2000 Genetic and Evolutionary Computation Conference Workshop Program, pp. 220–222.
[30] B.-T. Zhang, G. Paass, H. Mühlenbein, Convergence properties of incremental Bayesian evolutionary algorithms with single Markov chains, in: Proceedings of the 2000 Congress on Evolutionary Computation, vol. 2, pp. 938–945.
[31] B.-T. Zhang, S.-Y. Shin, Bayesian evolutionary optimization using Helmholtz machines, in: Parallel Problem Solving from Nature, Lecture Notes in Computer Science, vol. 1917, Springer, Berlin, 2000, pp. 827–836.

Byoung-Tak Zhang is an associate professor of Computer Science and Engineering at Seoul National University (SNU). He received his BS and MS degrees in computer engineering from SNU in 1986 and 1988, respectively, and a PhD in Computer Science from the University of Bonn, Germany, in 1992. Prior to joining SNU, Dr. Zhang has been a research associate at the German National Research Center for Information Technology (GMD). He serves as an associate editor of IEEE Transactions on Evolutionary Computation. His research interests are in learning and adaptive systems, evolutionary computation, probabilistic neural networks, and their application to real-world AI problems.



Dong-Yeon Cho is a PhD student in the School of Computer Science and Engineering, Seoul National University. He received his BS and MS degrees in computer engineering from Seoul National University in 1988 and 2000, respectively. His research interests include evolutionary computation, neural networks, and their probabilistic modeling.