A Unified Bayesian Framework for Evolutionary Learning and Optimization

Byoung-Tak Zhang[1]

Biointelligence Laboratory
School of Computer Science and Engineering
Seoul National University
Seoul 151-742, Korea
btzhang@bi.snu.ac.kr

Abstract. A probabilistic evolutionary framework is presented and shown to be applicable to both learning and optimization problems. In this framework, evolutionary computation is viewed as Bayesian inference that iteratively updates the posterior distribution of a population from the prior knowledge and observation of new individuals to find an individual with the maximum posterior probability. Theoretical foundations of Bayesian evolutionary computation are given and its generality is demonstrated by showing specific Bayesian evolutionary algorithms for learning and optimization. We also discuss how the probabilistic framework can be used to develop novel evolutionary algorithms that embed evolutionary learning for evolutionary optimization and vice versa.

1 Introduction

A number of evolutionary algorithms have recently been proposed that explicitly model the population of good solutions and use the constructed model to guide further search [2, 7, 19, 18]. These methods are generally known as the estimation of distribution algorithms or EDAs [17]. They use global information contained in the population, instead of using local information through crossover or mutation of individuals. From the population, statistics of the hidden structure are derived and used when generating new individuals.

Several methods have been proposed to build and sample from the distribution of the population. Baluja and Caruana propose the population-based incremental learning (PBIL) method that uses a single probability vector to replace the population [1]. The components of the vector are regarded independently of each other, so PBIL only takes into account first-order statistics. Mühlenbein and Paaß [16] present a univariate marginal distribution algorithm (UMDA) that estimates the distribution using univariate marginal frequencies in the set of selected parents, and resamples the new points. UMDA shows good performance on linear problems.

To capture more complex dependency, structures that can express higher-order statistics are necessary. De Bonet et al. [7] suggest a second-order method, called MIMIC (mutual information maximizing input clustring). It uses a chain structure to express conditional probabilities. Baluja and Davies [2] propose to use dependency trees to learn second-order probability distributions. Pelikan and Mühlenbein [19] suggest the bivariate marginal distribution algorithm (BMDA) as an extension of the UMDA. Mühlenbein and Mahnig [18] present the factorized distribution algorithm (FDA). Here, the distribution is decomposed into various factors or conditional probabilities. Pelikan et al. [20] describe the Bayesian optimization algorithm (BOA) that uses Bayesian networks in order to estimate the joint distribution of promising solutions. All these methods are designed for optimization.

Zhang [23] presents Bayesian evolutionary algorithms (BEAs) where evolutionary computation is formulated as a probabilistic process of finding an individual with the maximum a posteriori probability (MAP). The BEA starts with a population of individuals drawn from the prior distribution, and iteratively generates a new population by estimating the posterior fitness distribution of parent individuals and then sampling from the distribution offspring individuals via variation and selection operators. Explicit modeling of fitness distributions in terms of probabilities and the generational transition by means of Bayes formula are two distinguishing features of BEAs from other evolutionary algorithms. This framework was presented originally in the context of evolutionary learning of models from given data, i.e. function approximation. This is contrasted with most of EDAs that were suggested for function optimization.

In this paper, we extend the Bayesian evolutionary framework to encompass probabilistic evolutionary algorithms for function optimization as well as those for function approximation. Optimization and learning problems are formally defined and the similarities and differences between them are identified. We then illustrate how the existing evolutionary approaches can be formulated as Bayesian evolutionary algorithms and present a unified Bayesian framework of evolutionary computation for learning and optimization. The generality of the framework is illustrated by showing two specific applications of BEAs. One is the Bayesian evolutionary algorithm that performs function optimization by learning the sample distribution using a probabilistic graphical model, i.e. Helmholtz machines [6]. The effectiveness of this method is demonstrated on a suite of GA-deceptive functions. A second example is the Bayesian evolutionary algorithm that learns neural trees for time series prediction. Explicit formulae for specifying the distributions are provided and the effectiveness and robustness of the method is demonstrated on the laser data.

The paper is organized as follows. In Section 2, we formally define the problems of function optimization and function approximation (learning) and examine their relationship. Section 3 presents the unified Bayesian framework of evolutionary computation for learning and optimization. Section 4 demonstrates the effectiveness of the Bayesian evolutionary algorithm for learning in the context of evolving neural trees. Section 5 illustrates the usefulness of the Bayesian evolutionary algorithms for solving function optimization. Section 6 concludes with some remarks on the implications of the Bayesian evolutionary approaches to learning and optimization, respectively and in combination.

2 Optimization and Learning

2.1 Optimization

The goal of function optimization is to find $\mathbf{x}^*$ that minimizes an objective function f:

$$\mathbf{x}^* = \underset{\mathbf{x} \in X}{\operatorname{argmin}} \; f(\mathbf{x}), \tag{1}$$

where X is the input space of f. Here, the objective function f is given as an equation (or the equivalent) and the fitness of any search points $\mathbf{x}$ can be evaluated directly from this function.

Evolutionary algorithms have been extensively used for function optimization [3, 12, 22]. The initial population X^0 of search points $\mathbf{x}_i$ are generated at random. The fitness values $f(\mathbf{x}_i)$ of the points are measured. Depending on these values, the next population X^1 of search points are generated by a variety of variation operators. Typically, these involve mutation and crossover. Mutation generates a new point by modifying the existing point. Crossover generates new points by combining two or more search points. New populations X^t are generated until the optimum point $\mathbf{x}^*$ is found or the maximum allowed generation is reached.

2.2 Learning

Learning or function approximation involves constructing models f_θ of a target function f given a set D of input-output pairs $(\mathbf{x}_c, f(\mathbf{x}_c))$, i.e., $D = \{(\mathbf{x}_c, f(\mathbf{x}_c)) \mid c = 1, ..., N\}$. Here $f(\mathbf{x}_c)$ is the observed output of the target function given the input $\mathbf{x}_c$. The objective is to find the model θ^*, i.e., the functional structure and associated parameters, whose output $f_{\theta^*}(\mathbf{x})$ best predicts the output $f(\mathbf{x})$ of the target function given an arbitrary input $\mathbf{x}$. Using the squared error criterion as the objective function, this can be formulated as a minimization problem [5]:

$$\theta^* = \underset{\theta \in \Theta}{\operatorname{argmin}} \left\{ \sum_{\mathbf{x}_c \in D} ||f_\theta(\mathbf{x}_c) - f(\mathbf{x}_c)||^2 \right\}, \tag{2}$$

where Θ is the space of all possible models in consideration and D is the data set available.

Evolutionary computation has been used from its inception for automatic induction of models for a given system or process. For example, L. Fogel used simulated evolution to induce finite state machines that predict a sequence of symbols from an environment [9]. Other authors have used different structures as models of target systems. These include machine language instructions [10], sets of if-then rules [14], neural networks [26], and many others. Recent development includes genetic programming [15, 4] where Lisp-like symbolic programs are evolved from training data.

2.3 Optimization vs. Learning

There are several differences between function approximation (learning) and function optimization. First, the objective function f in optimization is given as an "explicit" functional form and the fitness of any search points $\mathbf{x}$ can be evaluated directly from this function. In contrast, the objective function in learning is given "implicitly" as a finite set D of function values, and thus the fitness of search points (in this case, models θ) is evaluated on this finite set of fitness cases.

Learning also differs from optimization in the way how the optimization result is used. The result θ^* of learning in (2) is usually used later to solve multiple problem-instances (e.g., to predict outputs $f_{\theta^*}(\mathbf{x})$ given different inputs $\mathbf{x}$) while the solution $\mathbf{x}^*$ of an optimization problem (1) is itself the final solution for the problem instance at hand.

Despite the differences in their goals and assumptions, function approximation is formally in close connection with 'regular' function optimization. To see the relationship, we define the objective function F of function approximation (2) in terms of D as

$$F(\theta) = \sum_{\mathbf{x}_c \in D} ||f_\theta(\mathbf{x}_c) - f(\mathbf{x}_c)||^2. \tag{3}$$

Then, the function approximation problem can be viewed as a function optimization problem:

$$\theta^* = \underset{\theta \in \Theta}{\operatorname{argmin}} \; F(\theta), \tag{4}$$

where Θ is the input space of the function F (see Equation (1) for comparison).

On the other hand, function optimization can be facilitated by using function approximation as a subroutine. That is, instead of attempting to directly optimize the objective function $f(\mathbf{x})$, we can build a model $f_\theta(\mathbf{x})$ of the objective function using the search points observed so far or their subset $X^t = \{\mathbf{x}_c \mid c = 1, ..., N\}$. The new problem is then formulated as:

$$\mathbf{x}^* = \underset{\mathbf{x} \in X}{\operatorname{argmin}} \; f_\theta(\mathbf{x}), \tag{5}$$

where f_θ is the approximated function of the original objective function f. This is reasonable since many optimization problems have underlying structure in their search space. Using this structure can help the search for the optimal solution.

Recently, a number of methods have been proposed that explicitly model, i.e. "learn", the population of promising points and use the constructed model to guide further search as reviewed in Section 1. These methods are generally known as the estimation of distribution algorithms or EDAs [17]. They use global information contained in the population, instead of using local information through crossover or mutation of individuals. From the population, statistics of the hidden structure are derived and used when generating new individuals.

In the following section, we present a general probabilistic framework for evolutionary computation that handles function optimization, function approximation, and their combinations in a uniform way.

3 A Bayesian Evolutionary Framework for Learning and Optimization

3.1 Bayesian Evolutionary Computation

In the Bayesian approach to evolutionary computation [23], the fitness of the individuals is formulated as a probability function. In the context of maximum likelihood, this might be a likelihood function or its transform. In decision theoretic contexts, this might involve complex loss functions. In its most general form, the fitness of an individual is represented as a posterior probability. Here, the best (fittest) individual is defined as the most probable model of the data with respect to the prior knowledge on the problem domain.

More formally, let θ denote the parameter vector for the model, let $\pi(\theta)$ be the prior probability distribution for the models (since θ uniquely determines the model, we use the terms 'model' and 'model parameter' interchangeably in this paper) and $p(D|\theta)$ the likelihood of the model for the data $D = \{(\mathbf{x}_c, y_c) \mid c = 1, ..., N\}$. Then, using Bayes formula [11] the posterior probability $\pi(\theta|D)$ of model θ is given as

$$\pi(\theta|D) = \frac{p(D|\theta)\pi(\theta)}{p(D)}. \tag{6}$$

Here, $p(D)$ is a normalizing constant and computed as

$$p(D) = \int_{\Theta} p(D|\theta)\pi(\theta)d\theta, \tag{7}$$

where Θ is the space of all possible model parameters (in case of discrete space, the integral will be replaced with a summation).

Initially, the shape of the (prior) probability distribution of individuals $\pi_0(\theta)$ is flat to reflect the fact that little is known at the outset (see Figure 1). Evolution is considered as an iterative process of revising the posterior distribution of individuals $\pi_t(\theta|D)$ by combining the prior $\pi_t(\theta)$ with the likelihood $p(D|\theta)$. In each generation, Bayes theorem (6) is used to estimate the posterior fitness of individuals from their prior fitness values. The posterior distribution $\pi_t(\theta|D)$ is then used to generate its offspring.

The aim of Bayesian evolutionary computation is twofold, depending on the problem to address. One is to choose a model θ_{MAP} that maximizes the posterior probability (MAP):

$$\theta_{MAP} = \underset{\theta \in \Theta}{\operatorname{argmax}}\ \pi(\theta|D). \tag{8}$$

The MAP model is then used to predict the output values y for given input values $\mathbf{x}$:

$$y = f(\mathbf{x}; \theta_{MAP}). \tag{9}$$

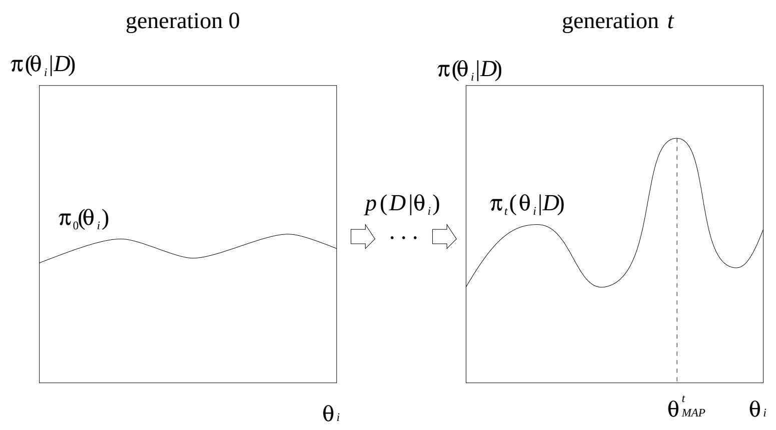

Fig. 1. Bayesian formulation of evolutionary computation. Initially, the shape of the (prior) probability distribution of models $\pi_0(\theta_i)$ is relatively flat, reflecting the fact that little is known at the outset. Evolution is considered as an iterative process of updating the posterior distribution of models $\pi_t(\theta_i|D)$ by combining the prior probability $\pi(\theta_i)$ with the likelihood $p(D|\theta_i)$ of observed data D.

This is the approach we take for function approximation or learning. An example of this approach will be given in Section 4 below.

Alternatively, the samples from the posterior distribution $\pi(\theta|D)$ can be used to compute the posterior *predictive* distribution $p(\mathbf{x}|D)$ of inputs $\mathbf{x}$ as follows:

$$p(\mathbf{x}|D) = \int p(\mathbf{x}, \theta|D)d\theta \tag{10}$$

$$= \int p(\mathbf{x}|\theta, D)\pi(\theta|D)d\theta \tag{11}$$

$$= \int p(\mathbf{x}|\theta)\pi(\theta|D)d\theta. \tag{12}$$

In the first two lines the distribution of future inputs $\mathbf{x}$ is expressed as an average of conditional predictions over the posterior distribution $\pi(\theta|D)$ of θ. The last equation above follows because $\mathbf{x}$ and D are conditionally independent given θ. This is the approach we take to generate promising new points for function optimization.

The distribution estimation algorithms for optimization mentioned in foregoing sections can be regarded as special cases of the Bayesian approach, where the maximum-likelihood estimate approximates the (full) posterior predictive distribution. That is, a single function $p(\mathbf{x}|\hat{\theta})$ of maximum likelihood estimate $\hat{\theta}$ is used to approximate the objective function $f(\mathbf{x})$ rather than using $p(\mathbf{x}|\theta)$'s for all possible θ's to take into account their distribution $\pi(\theta|D)$. Note that this is a reasonable approximation since if the prior is relatively flat and the peak of the likelihood function is relatively sharp, then the integral in (12) will be dominated by the region around the maximum likelihood estimate $\hat{\theta}$, and the posterior predictive distribution can be given approximately by

$$p(\mathbf{x}|D) \approx p(\mathbf{x}|\hat{\theta}) \int \pi(\theta|D) d\theta \tag{13}$$

$$= p(\mathbf{x}|\hat{\theta}), \tag{14}$$

where we used $\int \pi(\theta|D)d\theta = 1$, and $\hat{\theta}$ is the parameter vector that maximizes $p(\mathbf{x}|\theta)$. In Section 5 below, we describe a method that uses a probabilistic graphical model known as Helmholtz machines to approximate the posterior predictive distribution function $p(\mathbf{x}|D)$ by the maximum likelihood $p(\mathbf{x}|\hat{\theta})$.

3.2 The Canonical Bayesian Evolutionary Algorithm

Given the principles of the Bayesian evolutionary approach to optimization and learning, we are now ready to describe their realizations. Algorithm 3.1 summarizes the canonical Bayesian evolutionary algorithm.

Algorithm 3.1 [Canonical BEA]

1. (Initialize) Generate $\Theta^0 = \{\theta_1^0, ..., \theta_M^0\}$ from the prior distribution $\pi_0(\theta)$. Initialize data size N_0 and temperature T_0. Set generation count $t \leftarrow 0$.
2. (D-step) Generate (observe) D^t of size N_t. Compute likelihoods $p(D^t|\theta_i^t)$.
3. (P-step) Estimate posterior distribution $\pi_t(\theta|D^t)$ of the individuals in Θ^t.
4. (V-step) Generate L variations $\Theta' = \{\theta_1', ..., \theta_L'\}$ by sampling from $\pi_t(\theta|D^t)$.
5. (S-step) Select M individuals from Θ' into $\Theta^t = \{\theta_1^{t+1}, ..., \theta_M^{t+1}\}$ based on $p(D^t|\theta_i')$. Set the best individual θ_{best}^t.
6. (R-step) Optionally, revise prior distribution $\pi_t(\theta)$, and update N_t and T_t.
7. (Loop) If the termination condition is met, then stop. Otherwise, set $t \leftarrow t+1$ and go to Step 2.

In essence, the algorithm consists of five steps: D (data), P (posterior), V (variation), S (selection), and R (revision). The three steps of R, D, and P involve computation of prior, likelihood, and posterior probabilities, respectively. The V and S steps realize the sampling from the posterior distribution. Note that BEAs attempt in the P-step to explicitly model the posterior fitness distribution $\pi_t(\theta|D^t)$ of individuals in population Θ^t. Another feature of BEAs is the D-step which may care for incremental growth of data sets [27]. This naturally corresponds to the Bayesian inductive learning principle. The R-step is used only for adaptive versions of BEAs, and will not be considered in this paper.

More specifically, we define the fitness value of individual θ_i^t as its posterior probability $\pi_t(\theta_i^t|D^t)$:

$$\pi_t(\theta_i^t|D^t) \equiv \frac{p(D^t|\theta_i^t)\pi_t(\theta_i^t)}{\sum_{\theta_j^t \in \Theta^t} p(D^t|\theta_j^t)\pi_t(\theta_j^t)}, \tag{15}$$

where Θ^t is the finite set of inidividuals in the tth population. Assuming the exponential family for the likelihood function and prior distribution (e.g., Gaussian distributions),

$$p(D^t|\theta_i^t) = \frac{1}{Z_E}\exp\{-E(D^t|\theta_i^t)/T_t\} \tag{16}$$

$$\pi_t(\theta_i^t) = \frac{1}{Z_C}\exp\{-C(\theta_i^t)/T_t\}, \tag{17}$$

the fitness of individuals is written as

$$\begin{aligned}\pi_t(\theta_i^t|D^t) &\equiv \frac{\exp\{-F(\theta_i^t|D^t)/T_t\}}{\sum_{\theta_j^t\in\Theta^t}\exp\{-F(\theta_j^t|D^t)/T_t\}} \\ &= \frac{\exp\{-(E(D^t|\theta_i^t)+C(\theta_i^t))/T_t\}}{\sum_{\theta_j^t\in\Theta^t}\exp\{-(E(D^t|\theta_j^t)+C(\theta_j^t))/T_t\}},\end{aligned} \tag{18}$$

where $E(D|\theta_j^t)$ and $C(\theta_j^t)$ are arbitrary component measures for evaluating raw fitness of individuals, and T_t is the temperature parameter for controlling the randomness of the stochastic process. Note here that the posterior probability is approximated by a fixed-size population Θ^t which is typically a small subset of the entire model space Θ: $\Theta^t \subset \Theta, |\Theta^t| \ll |\Theta|$. The evolutionary inference step from generation t to $t+1$ is then considered to induce a new fitness distribution $\pi_{t+1}(\theta)$ from $\pi_t(\theta)$ following Bayes formula.

At each generation t we keep the best individual θ_{best}^t which is the individual with the maximum a posteriori (MAP) probability with respect to Θ_t:

$$\begin{aligned}\theta_{best}^t &= \operatorname*{argmax}_{\theta_i^t} \pi_t(\theta_i^t|D^t) &(19)\\ &= \operatorname*{argmax}_{\theta_i^t} \frac{p(D^t|\theta_i^t)\pi_t(\theta_i^t)}{\sum_{\theta_j^t} p(D^t|\theta_j^t)\pi_t(\theta_j^t)} \\ &= \operatorname*{argmax}_{\theta_i^t} p(D^t|\theta_i^t)\pi_t(\theta_i^t), &(20)\end{aligned}$$

where θ_i^t and θ_j^t are elements of population Θ^t. A complete run for t generations of the Bayesian evolutionary algorithm then chooses the best among the generation-best models, i.e., $\theta_{best}(t)$ such that

$$\pi_t(\theta_{best}(t)|D^t) = \max_{k\le t} \pi_k(\theta_{best}^k|D^t), \tag{21}$$

where θ_{best}^k is the best solution at generation k and $\pi_t(\theta_{best}(t)|D^t)$ is the tth estimation of $\pi(\theta_{MAP}|D^t)$.

Note that the description of Algorithm 3.2 is intentionally abstract and general. Thus, for example, the V-step can be implemented in several ways, including mutation, crossover, Metropolis-Hastings moves, or their combinations. The S-step can also be realized using various selection schemes, such as truncation selection and tournament selection as well as proportional selection [3].

4 Bayesian Evolutionary Computation for Learning

The Bayesian evolutionary algorithm is applied to learning neural tree models of time series data. We provide explicit probability models for the prior, likelihood, and posterior probability distributions of neural trees. The sampling procedure for evolving neural trees are described and empirical results are reported.

4.1 Definining Probability Distributions of Neural Trees

Neural trees are tree-structured neural networks. A neural tree is composed of terminal nodes, nonterminal nodes, and the weights of connection links between two nodes [26]. The nonterminal nodes represent neural units with various activation functions. Typical neurons compute the sum of weighted inputs from the lower layer by

$$net_i = \sum_j w_{ij} y_j, \tag{22}$$

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

$$y_i = f(net_i) = \frac{1}{1 + e^{-net_i}}. \tag{23}$$

In principle, there is no restriction in the types of activation functions employed in neural trees since the evolutionary learning procedure does not impose limiting constraints such as continuity or differentiability on the search space.

Neural trees have a number of important features from the model induction point of view. Neural trees can represent a broad class of higher-order networks since a single tree can have a mixture of sigma, pi, and any other types of units. No bound is enforced in the number of layers of the network. The network structures are not strictly layered; the connections between non-neighboring layers are allowed. The network may contain partial connectivity, which is useful for the economic representation of arbitrary complex interactions. In addition, neural trees do not require decoding for their fitness evaluation. Training and fitness evaluation can be performed directly on the genotype since both the genotype and phenotype are equivalent.

To learn the fittest model by the BEAs, we first define the probability distributions of neural tree models for data. A neural tree model is parameterized as $\theta = (\mathbf{w}, k)$, where k is the number of nodes in the neural tree (including bias terms) and $\mathbf{w}$ is the weight vector. The posterior probability of a neural tree θ is written as

$$\begin{aligned} \pi(\theta|D) &\propto p(D|\theta)\pi(\theta) & (24) \\ &= p(D|\mathbf{w}, k)\pi(\mathbf{w}, k) & (25) \\ &= p(D|\mathbf{w}, k)\pi(\mathbf{w}|k)\pi(k). & (26) \end{aligned}$$

Given the training data

$$D = \{(\mathbf{x}_c, y_c) \mid c = 1, ..., N\}, \tag{27}$$

the model A can represent the following input-output mapping

$$y_c = f(\mathbf{x}_c; \theta) + \epsilon. \tag{28}$$

Here, the noise ϵ is assumed to be Gaussian with mean zero and standard deviation σ.

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

$$\begin{aligned} p(D|\mathbf{w}, k) &= \prod_{c=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\Big(-\frac{(y_c - f(\mathbf{x}_c; \mathbf{w}, k))^2}{2\sigma^2}\Big) \\ &= \Big(\frac{1}{\sqrt{2\pi}\sigma}\Big)^N \exp\Big(-\frac{\sum_{c=1}^{N}(y_c - f(\mathbf{x}_c; \mathbf{w}, k))^2}{2\sigma^2}\Big). \end{aligned} \tag{29}$$

We define the following prior probability for weights of the neural tree

$$\begin{aligned} \pi(\mathbf{w}|k) &= \prod_{j=1}^{k-1} \frac{1}{\sqrt{2\pi}} \exp\Big(-\frac{w_j^2}{2}\Big) \\ &= \Big(\frac{1}{\sqrt{2\pi}}\Big)^{k-1} \exp\Big(-\frac{\sum_{j=1}^{k-1} w_j^2}{2}\Big), \end{aligned} \tag{30}$$

where the components of the weight vector $\mathbf{w}$ are assumed to be independent of each other and distributed according to zero-mean Gaussian with standard deviation 1. We also assume that the number of nodes in the neural tree is distributed according to following Poisson distribution

$$\pi(k-3) = \frac{\lambda^{k-3}\exp(-\lambda)}{(k-3)!}, \tag{31}$$

where $k = 3, 4, ...$ since the neural tree which consists of one terminal node was not considered. Substituting Equations (29), (30), and (31) into (24), we obtain the following posterior probability for the neural tree

$$\begin{aligned} \pi(\theta|D) &\propto p(D|\mathbf{w}, k)\pi(\mathbf{w}|k)\pi(k) \\ &= \Big(\frac{1}{\sqrt{2\pi}\sigma}\Big)^N \exp\Big(-\frac{\sum_{c=1}^{N}(y_c - f(\mathbf{x}_c; \mathbf{w}, k))^2}{2\sigma^2}\Big) \\ &\times \Big(\frac{1}{\sqrt{2\pi}}\Big)^{k-1} \exp\Big(-\frac{\sum_{j=1}^{k-1} w_j^2}{2}\Big) \frac{\lambda^{k-3}\exp(-\lambda)}{(k-3)!}, \end{aligned} \tag{32}$$

where $\theta = (\mathbf{w}, k)$ is the parameter vector for the neural tree.

4.2 The BEA for Learning

The general procedure for evolving neural trees is similar to Algorithm 3.2. It consists of four major steps: D (data), P (posterior), V (variation), and S (selection). The D and P steps involve computation of the likelihood $p(D|\theta_i^t)$ and posterior probabilities $\pi_t(\theta|D)$, respectively. The V and S steps realize the sampling from the posterior distribution. The probability models for the component distributions are as given in the previous subsection.

To search the structure and parameters of neural trees, we maintain a population Θ^t of individuals θ_i at tth generation

$$\Theta^t = \{\theta_1, \theta_2, ..., \theta_M\}, \tag{33}$$

where M is the population size. The initial population Θ^0 is created according to the prior probability of models. Each individual $\theta_i = (\mathbf{w}_i, k_i)$ is generated by sampling first a value k_i for the number of nodes from the Poisson distribution (31) and then $\mathbf{w}_i$ from the Gaussian distribution (30) using the k_i-value. In each generation t, the error $E_i(t)$ of neural trees is measured on the training set D as

$$E_i(g) = \sum_{c=1}^{N} (y_c - f(\mathbf{x}_c; \theta_i))^2, \tag{34}$$

where $f(\mathbf{x}_c; \theta_i)$ is the actual output for input vector $\mathbf{x}_c$ of neural tree θ_i, and y_c is the target output. Using this value, the likelihood (29) of the neural tree is computed. Finally, the posterior probability of each model is computed by Equation (32).

```
1. (Initialize) Set the training set $D = \{(\mathbf{x}_c, y_c) \mid c = 1, ..., N\}$. Generate $\Theta^0 = \{\theta_1^0, ..., \theta_M^0\}$ from $\pi_0(\theta)$. Set generation count $t \leftarrow 0$.
2. (D-step) Compute likelihoods $p(D|\theta_i^t)$ of individuals $\theta_i^t = (\mathbf{w}_i^t, k_i^t)$.
3. (P-step) Estimate posterior distribution $\pi_t(\theta|D)$ of the individuals in $\Theta^t$.
4. (V-step) Generate $L$ variations $\Theta' = \{\theta_1', ..., \theta_L'\}$ by sampling from $\pi_t(\theta|D)$. Optionally, train the weights $\mathbf{w}$ of the individuals $\theta$.
5. (S-step) Select $M$ individuals from $\Theta'$ into $\Theta^t = \{\theta_1^{t+1}, ..., \theta_M^{t+1}\}$ based on $p(D|\theta_i')$. Set the best individual $\theta_{best}^t$.
6. (Loop) If the termination condition is met, then stop. Otherwise, set $t \leftarrow t + 1$ and go to Step 2.
```

Fig. 2. Outline of the Bayesian evolutionary algorithm for learning neural trees.

To construct the next generation Θ^{t+1}, a candidate tree θ_i' is first created from the parent tree θ_i in the current population. The candidate tree is then accepted with probability

$$\alpha(\theta_i, \theta_i') = \min\left\{1, \frac{\pi(\theta_i'|D)}{\pi(\theta_i|D)}\right\}, \tag{35}$$

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

Two major variation operators are applied to the parent models for generating candidate models. First, crossover operators swap two subtrees chosen at random from the parent tree θ_i and another tree $\theta_j, (i \neq j)$ which is selected randomly from the current population to create the candidate model θ_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. The probabilities for applying these operators are p_c and p_m, respectively. These mating steps are performed iteratively until L individuals are produced.

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 a Gaussian mutation expression

$$w_{ij}' = w_{ij} + \mathrm{N}(0,1) \qquad j = 1, 2, ..., k_i - 1, \tag{36}$$

where k_i is the number of nodes in tree θ_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 Equation (35).

The offspring population Θ' is obtained through the above procedure and we finally generate the parent population Θ^{t+1} of the next generation by selecting the best M individuals from Θ'.

4.3 Empirical Results

To see the potential advantages of probabilistic approach underlying the BEAs for learning, we carried out experiments on time series data. The far-infrared NH_3 laser data was chosen as a test problem [26]. We used the first 500 data points for evolving the neural tree models and the rest 500 data points for testing the predictive accuracy. Other experimental setup is as follows: the maximum number of branches of a nonterminal node is 3, which is the same as the input size, the standard deviation of the noise is $\sigma = 0.05$, and the maximum number of fitness evaluations is 10^6. The candidate population size is identical to the parent population size ($M = L$) in all experiments.

The effects of priors were analyzed by running BEAs with different mean values of $\lambda = 10, 20, 30, 40$ for the Poisson distribution on the number of nodes. Figures 3 and 4 show the effect of priors on the complexities of the best neural trees in terms of the number of nodes and the squared sum of weight values.

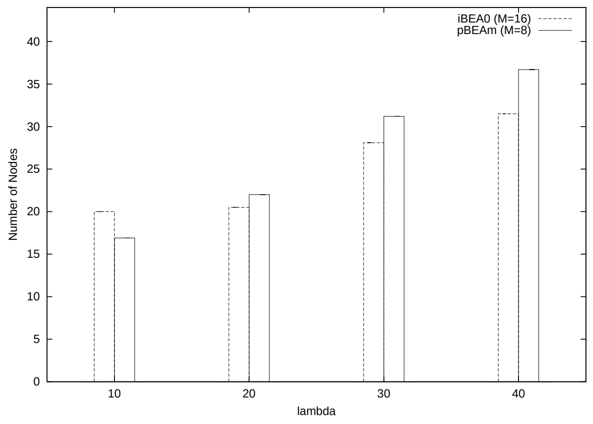

Fig. 3. Effects of the priors on the number of nodes in the solution trees for the laser problem.

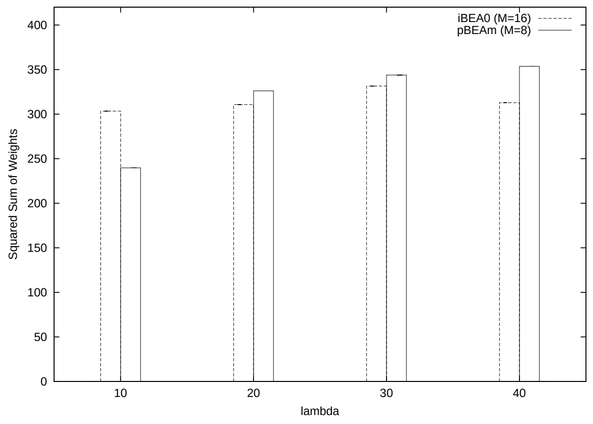

Fig. 4. Effects of the priors on the weight size in the solution trees for the laser problem.

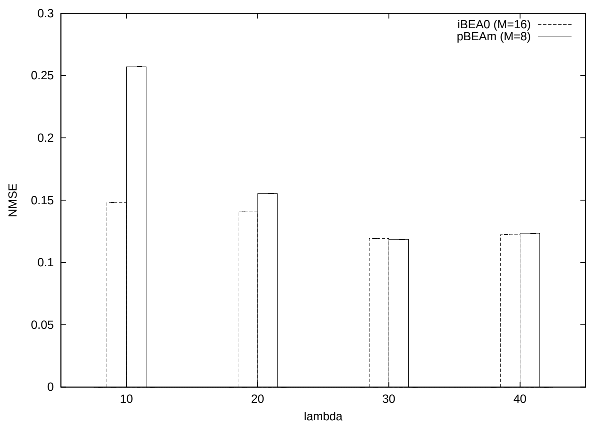

Fig. 5. Effects of the priors on the predictive accuracy of solution trees for the laser problem.

The corresponding performance (in normalized mean squared error) of the neural trees are shown in Figure 5. To take into account the variation caused by algorithm parameters, we ran two different versions of BEAs with varying population sizes and variation operators. Ten runs were made for each algorithm with each parameter setting.

The results show that for $\lambda = 20$ and 30 the size of solution trees is approximately the same as λ, while for $\lambda = 10$ the solution size was around 18 and for $\lambda = 40$ the solution size was around 35. This suggests that the size of best solutions for this problem is likely to lie between 20 and 35. This reasoning is confirmed by seeing the accuracy given in Figure 5, which indicates the best solution size of 30 nodes. The results also show that the prior effect is relatively robust in the sense that incorrect priors, as in the case of $\lambda = 20$, can still lead to acceptable solutions. The results for $\lambda = 40$ indirectly show the Occam's razor effect (i.e. parsimony pressure) of priors: unnecessary complexity is avoided. Figure 4 shows that weight values are less strongly dependent on λ than the number of nodes. This means that harmful overfitting, at least serious one, did not occur. One possible explanation for this is that local search is better guided by the probability distribution models. Another reason might be the Occam's razor effect, that is, preferring parsimonious solutions avoids overfitting to the data.

To summerize, the experimental results support the effectiveness of priors in guiding the evolutionary learning of tree structures and weights; good priors facilitate the search process and result in better solutions. This is contrasted with conventional evolutionary algorithms where it is usually difficult to incorporate the prior knowledge about the problem domain.

5 Bayesian Evolutoinary Computation for Optimization

In this section, a method for Bayesian evolutionary optimization is presented. It is based on the distribution estimation of the sample population. To find the $\mathbf{x}$-point that maximizes the objective function f, this method builds a model f_θ of f using the current population of search points. We present a distribution estimation method that uses the Helmholtz machine [6], a probabilistic graphical model.

5.1 The BEA for Optimization

The algorithm is summarized in Figure 6. Initially, a population X^0 of M search points $\mathbf{x}_i^0$, $i = 1, ..., M$, are generated from a prior distribution $\pi_0(\mathbf{x})$. It should be stressed that the search space we are here concerned with is the $\mathbf{x}$-space, not the θ-space. Note also that the population in this optimization context is denoted as X^t, not D which is reserved to denote the training data set in the context of learning. Once created, the fitness values of the points are evaluated and their likelihoods $p(X^t|\theta)$ are computed, where θ is the parameter vector for the probability model of the population. Combining the prior $\pi_0(\mathbf{x})$ and likelihood $p(X^t|\theta)$, the posterior probability $\pi(\theta|X^t)$ of individuals is computed using the Bayes rule:

$$\pi(\theta|X^t) = \frac{p(X^t|\theta)\pi(\theta)}{p(X^t)}. \tag{37}$$

Since $p(X^t)$ does not depend on the parameter vector θ, maximization of Equation (37) is equivalent to maximizing the numerator, i.e.

$$\pi(\theta|X^t) \propto p(X^t|\theta)\pi(\theta). \tag{38}$$

Note that, under the uniform prior for θ, the maximization of Equation (38) is reduced to finding the maximum likelihood estimate $\hat{\theta}$:

$$\theta^* = \arg\max_\theta \pi(\theta|X^t) = \arg\max_\theta p(X^t|\theta) = \hat{\theta}. \tag{39}$$

Thus, the posterior distribution $\pi(\theta|X^t)$ can be approximated by the maximum likelihood $p(X^t|\hat{\theta})$. We make use of this equivalence and use a Helmholtz machine to find $p(X^t|\hat{\theta})$.

Once the posterior distribution (or its MLE approximation) is estimated, new points are generated by sampling from the posterior *predictive* distribution $p(\mathbf{x}|X^t)$. To do this, we make use of the following relationship

$$p(\mathbf{x}|X^t) \approx p(\mathbf{x}|\hat{\theta}) \int \pi(\theta|X^t) d\theta \tag{40}$$

$$= p(\mathbf{x}|\hat{\theta}), \tag{41}$$

which follows from the similar arguments given for Equation (14).

```
1. (Initialize) Generate $X^0 = \{\mathbf{x}_i^0, ..., \mathbf{x}_M^0\}$ from the prior distribution $\pi_0(\mathbf{x})$. Set generation count $t \leftarrow 0$.
2. (D-step) Given $X^t$, compute the parameter $\hat{\theta}$ that maximizes the likelihood $p(X^t|\theta)$ using a Helmholtz machine.
3. (P-step) Estimate the posterior predictive distribution $p_{t+1}(\mathbf{x}) = p(\mathbf{x}|X^t) \approx p(\mathbf{x}|\hat{\theta})$ using the Helmholtz machine.
4. (V-step) Generate $L$ variations $X' = \{\mathbf{x}'_1, ..., \mathbf{x}'_L\}$ by sampling from the distribution $p_{t+1}(\mathbf{x})$ using $\hat{\theta}$ of the Helmholtz machine.
5. (S-step) Select $M$ best points from $X'$ and $X^t$ into $X^{t+1} = \{\mathbf{x}_1^{t+1}, ..., \mathbf{x}_M^{t+1}\}$ based on their fitness values $f(\mathbf{x})$.
6. (Loop) If the termination condition is met, then stop. Otherwise, set $t \leftarrow t + 1$ and go to Step 2.
```

Fig. 6. Outline of the Bayesian evolutionary algorithm for optimization where the Helmholtz machine is used for density estimation.

To summarize, the BEA for optimization consists of four main steps: data observation (D), probability estimation (P), variation (V), and selection (S) steps. In the P-step, the density of the current population X^t is estimated, in this case, by a Helmholtz machine [6]. Since the samples may not be very representative of the distribution, especially in early generations, the weights of the Helmholtz machine are reinitialized each generation. This avoids trapping in local minima. In the V-step, the learned Helmholtz machine is used to generate offspring population X' of L data points. More details on learning and simulating from the Helmholtz machine can be found in [28]. In the S-step, M best points are chosen into the next population X^{t+1} from the union of X^t and X'. In the experiments, we use $L = M$. This is similar to the $(\mu + \lambda)$ evolution strategy [3] with $\mu = \lambda = M$.

5.2 Empirical Results

Experiments have been performed on a suite of deceptive functions from the literature [19, 20]. We used the one-max function, quadratic function, 3-deceptive function, and trap-5 function. Here we report the results of the first two problems; additional results can be found in [28].

The performance of the Helmholtz machine was compared with that of the simple genetic algorithm (sGA). The sGA used one cut-point crossover, one point mutation, and roulette-wheel selection. The parameters for sGA were: maximum generation = 10^7, population size = 50, crossover rate = 0.5, and mutation rate = 0.01. The parameters of the BEA with the Helmholtz machine were: maximum generation = 10^5, population size = 50, learning rate = 0.5, and number of iterations = 1000.

Tables 1 and 2 summarize the results for the one-max and quadratic functions, respectively. The results shown are average values over 10 runs. The algorithm is considered as converged if the population contains over 10% of optimal solutions. The entry of the tables marked with '-' means that none of the runs found an optimal solution. The results show that the Bayesian evolutionary algorithms with the Helmholtz machine outperform the simple genetic algorithm both in the success rate and in the number of iterations. BEAs find the optimal solutions all the time while the simple GA finds solutions for small size problems only. It is also interesting that the BEAs find solutions much faster than the sGAs when both find optimal solutions. The table shows that the computational time (in the number of interations) for sGAs grow exponentially while that for the BEAs grow almost linearly.

Table 1. Results on the one-max function for the Helmholtz machine algorithm (HM) and the simple genetic algorithm (sGA).

Problem	Success %		Number of Iterations	
Size	HM	sGA	HM	sGA
20	100	100	6.3	930.7
40	100	100	34.7	16,904.7
60	100	50	175.4	5,691,291
80	100	0	657.7	-
100	100	0	2999.3	-

Table 2. Results on the quadratic function for the Helmholtz machine algorithm (HM) and the simple genetic algorithm (sGA).

Problem	Success %		Number of Iterations	
Size	HM	sGA	HM	sGA
20	100	100	5.9	12,073.3
40	100	10	33.0	6,532,588
60	100	0	198.0	-
80	100	0	765.6	-
100	100	0	3430.7	-

6 Concluding Remarks

We presented a unified probabilistic framework for solving optimization and learning problems using evolutionary computation. In this framework, evolutionary computation is formulated as a probabilistic process of Bayesian inference, which leads to a class of Bayesian evolutionary algorithms or BEAs. The generality and concreteness of the framework is demonstrated by showing two specific instances of BEAs. One is for optimization and the other for learning.

The method of Bayesian evolutionary optimization is very closely related to the distribution estimation algorithms. Similar to the usual EDAs, a BEA builds a probabilistic model of discrete samples. However, BEAs are more general in the sense that the distribution estimated is the posterior distribution which encompasses likelihood functions (as in most EDAs) as a special case. We have shown that graphical models such as Helmholtz machines provide a powerful tool for implementing the BEA for optimization. We also show that the learning problems can be effectively solved by adopting the Bayesian evolutionary approach. This was shown in the context of evolving neural tree models for predicting time series data. The probabilistic approach allows the evolution to be guided more effectively by using principled specification of prior knowledge.

It is interesting to see that learning helps solve optimization problems (as in the distribution estimation algorithms), while optimization helps solve learning problems (as in genetic programming and other evolutionary algorithms for machine learning). In both cases, evolutionary computation can be used as a tool for learning and optimization. In this work, we studied two rather loosely coupled combinations of learning and optimization. However, it is not difficult to imagine more tightly and cross-coupled combinations of evolutionary learning and optimization. For example, it would be interesting to have an evolutionary algorithm that "learns" the distribution of a training data set using evolutionary "optimization" where the training data set itself is collected incrementally by an evolutionary "learning" process to "optimize" some information criterion.

Acknowledgments

This research was supported by the Korea Science and Engineering Foundation (KOSEF) under Grant 981-0920-350-2, 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 the BK21 Program.

References

1. Baluja, S., Caruana, R.: Removing the genetics from the standard genetic algorithm. Technical Report CMU-CS-95-141, Carnegie Mellon University (1995)
2. Baluja, S., Davies, S.: Using optimal dependency-trees for combinatorial optimization: learning the structure of the search space. In *Proc. 14th Int. Conf. on Machine Learning*, Morgan-Kaufmann, (1997) 30–38

3. Bäck, T.: *Evolutionary Algorithms in Theory and Practice.* Oxford Univ. Press (1996)
4. Banzhaf, W., Nordin, P., Francone, F.: *Genetic Programming: An Introduction.* San Mateo, CA: Morgan Kaufmann (1998)
5. Cherkassky, V., Mulier, F.: *Learning from Data: Concepts, Theory, and Methods.* Wiley (1998)
6. Dayan, P., Neal, G.E., Zemel, R.S.: The Helmholtz machine. *Neural Computation*, Vol. 7. (1995) 1022–1037
7. De Bonet, J.S., Isbell, C.L., Viola, P.: MIMIC: Finding optima by estimating probability densities. In *Advances in Neural Information Processing Systems 9*, The MIT Press (1997) 424–430
8. Fogel, D.B., A. Ghozeil, A.: Using fitness distributions to design more efficient evolutionary computations. In *Proc. 1996 IEEE Int. Conf. on Evolutionary Computation*, Piscataway, NJ: IEEE Press (1996) 11–19
9. Fogel, D.B. (ed.): *Evolutionary Computation: The Fossil Record.* IEEE Press (1998)
10. Friedberg, R. M.: A learning machine: Part I, *IBM J. Research and Development*, Vol. 2. No. 1. (1958) 2–13
11. Gelman, A., Carlin, J.B., Stern, H.S., Rubin, D.B.: *Bayesian Data Analysis*, London: Chapman & Hall (1995)
12. Goldberg, D.E.: *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley (1989)
13. Hinton, G.E., Dayan, P., Frey, B.J., Neal R. M.: The wake-sleep algorithm for unsupervised neural networks. *Science*, Vol. 268. (1995) 1158–1160
14. Holland, J. H.: Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In *Machine Learning: An Artificial Intelligence Approach*, Vol. II, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, Eds. San Francisco: Morgan Kaufmann, (1986) 593–623
15. Koza, J. R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, MA: MIT Press (1992)
16. Mühlenbein, H., Paaß, G.: From recombination of genes to the estimation of distributions I: Binary parameters. In *Parallel Problem Solving from Nature IV*, LNCS 1141, Springer (1996) 178–187
17. Mühlenbein, H., Mahnig, T., A. Ochoa: Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, Vol. 5. (1999) 215–247
18. Mühlenbein, H., Mahnig, T.: FDA - A scalable evolutionary algortihm for the optimization of additively decomposed functions. *Evolutionary Computation*, Vol. 7. No. 4. (1999) 353–376
19. Pelikan, M., Mühlenbein, H.: The bivariate marginal distribution algorithm. *Advances in Soft Computing - Engineering Design and Manufacturing*, London: Springer-Verlag (1999) 521–535
20. Pelikan, M., Goldberg, D.E., Cantú-Paz, E.: BOA: The Bayesian optimization algorithm. In *Proc. 1999 Genetic and Evolutionary Computation Conference*, CA: Morgan Kaufmann (1999) 525–532
21. Rissanen, J.: Stochastic complexity and modeling. *The Annals of Statistics*, Vol. 14. (1986) 1080–1100.
22. Schwefel, H.-P.: *Evolution and Optimum Seeking*, Sixth Generation Computer Technology Series, Wiley, New York (1995)
23. Zhang, B.-T.: A Bayesian framework for evolutionary computation. In *Proc. 1999 Congress on Evolutionary Computation* (CEC99), IEEE Press (1999) 722–727
24. Zhang, B.-T.: Bayesian methods for efficient genetic programming. *Genetic Programming and Evolvable Machines*, Vol. 1. No. 3. (2000) 217–242

25. Zhang, B.-T., Cho, D.-Y.: Evolving neural trees for time series prediction using Bayesian evolutionary algorithms, In *Proc. First IEEE Workshop on Combinations of Evolutionary Computation and Neural Networks* (ECNN-2000), IEEE Press (2000) 17-23
26. Zhang, B.-T., Ohm, P., Mühlenbein, H.: Evolutionary induction of sparse neural trees. *Evolutionary Computation*, Vol. 5. No. 1. (1997) 213–236
27. Zhang, B.-T., Paaß, G., Mühlenbein, H.: Convergence properties of incremental Bayesian evolutionary algorithms with single Markov chains. *Congress on Evolutionary Computation* (CEC-2000), IEEE Press (2000) 938-945
28. Zhang, B.-T., Shin, S.-Y.: Bayesian evolutionary optimization using Helmholtz machines, *Int. Conf. on Parallel Problem Solving from Nature* (PPSN-2000), Vol. 1917. (2000) 827–836