Bayesian Evolutionary Algorithms for Continuous Function Optimization

Soo-Yong Shin
Artificial Intelligence Lab (SCAI)
School of Computer Science and Engineering
Seoul National University
Seoul 151-742, Korea.
syshin@scai.snu.ac.kr

Byoung-Tak Zhang
Artificial Intelligence Lab (SCAI)
School of Computer Science and Engineering
Seoul National University
Seoul 151-742, Korea.
btzhang@cse.snu.ac.kr

Abstract- Recently many researchers have studied the estimation of distribution algorithms (EDAs) as an optimization method. While most EDAs focus on solving combinatorial optimization problems, only a few algorithms have been proposed for continuous function optimization. In previous work, we developed a Bayesian evolutionary algorithm (BEA) for combinatorial optimization problem using a probabilistic graphical model known as Helmholtz machine. Since BEA is a general framework for evolutionary computation based on the Bayesian inductive principle, we improved BEA for continuous function optimization problems. By the nature of neural network and availability of the wake-sleep learning algorithm, Helmholtz machine can capture the continuous distribution with a small modification. The proposed method has been applied to a suite of benchmark functions and compared with a real-coded genetic algorithm and previous experimental results.

1 Introduction

Recently, a number of algorithms known as the estimation of distribution algorithms (EDA) have been proposed that explicitly model the population of good solutions and use the constructed model to guide further search[6, 13]. Instead of using local information through crossover or mutation of individuals, they use global information contained in the population. From the population, statistics of the hidden structure are derived and used for generating new individuals.

While most of the studies focus on building the model for discrete space, a few algorithms have been proposed for continuous spaces[14, 15, 17, 16, 7, 2]. By the way, most of continuous space approaches consider the joint density function as a product of unidimensional and independent normal densities, so they suffered from the lack of abilities of their model.

In this paper, we present a Bayesian evolutionary algorithm (BEA) that estimates the continuous distribution by a graphical learning model known as Helmholtz machine. In the previous work, we focused on evolutionary optimization and developed the Bayesian evolutionary algorithm for combinatorial optimization problems[23], where Helmholtz machine was used as a tool to capture the distribution of a given data set. A Bayesian evolutionary algorithm is a probabilistic model of evolutionary computation that is based on Bayesian inference[20, 22]. In BEA, evolutionary computation is formulated as a probabilistic process of finding an individual with maximum a posteriori probability (MAP). Though originally developed as a method for evolutionary learning, BEA can be shown to be applicable to evolutionary optimization as well. Since the BEA is a general framework, it can be used in continuous function optimization without modification. And Helmholtz machine is also easily expanded to continuous space, because Helmholtz machine is a kind of multilayer neural network[3].

The paper is organized as follows. In Section 2, previous works for capturing continuous distribution are reviewed. Section 3 presents Bayesian evolutionary algorithm framework. Section 4 describes the architecture and learning algorithm of Helmholtz machine and modification of Helmholtz machine for continuous space. Section 5 reports the experimental results. Conclusions are drawn in Section 6.

2 Previous works for continuous space modelling

In the early research that the EDA approach are proposed for optimization in continuous space, it is assumed that the joint density function follows an n-dimensional normal distribution which is factorized by a product of unidimensional and independent normal density[6, 7, 13]. So these approaches use simple models that do not cover any interactions in the problem.

In the Stochastic Hill-Climbing with Learning by Vectors of Normal Distribution (SHCLVND), Rudlof and Köppen[14] estimate the joint density function as a product of unidimensional and independent normal densities. The population of solutions is replaced and modelled by a vector of mean values of Gaussian normal distribution μ_i for each optimized variable. The standard deviation σ is stored globally and it is the same for all variables. After generating a number of new solutions, the mean values μ_i are adapted by means of the Hebbian rule and shifted towards the best of the generated solutions and the standard deviation σ is reduced to make future exploration of the search space narrower.

Sebag and Ducoulombie[16] propose an extension of the boolean PBIL algorithm to continuous space version, named PBILc. PBILc also assumed the joint density func-

0-7803-6657-3/01/$10.00 ©2001 IEEE

tion as unidimensional and independent normal densities like SHCLVND but various ways of modifying the σ parameter have been exploited in PBILc. They proposed four heuristics: (1) use a constant value for all the marginal and all the generations; (2) adjust it as in a (1, λ) ES; (3) calculate the sample variance of the K better individuals of each generation; (4) by means of a Hebbian rule.

In another implementation of a real-coded PBIL[17], an interval (a_i, b_i) and a number z_i are stored for each variable. If z_i stands for a probability of a solution to be in the right half of the interval, the best solutions are selected and the numbers z_i is shifted towards them. When z_i for a variable gets close to either 0 or 1, the interval is reduced to the corresponding half of it.

Salustowicz and Schmidhuber[15] suggest new algorithm where computer programs or mathematical functions are evolved as in genetic programming. In the probabilistic incremental program evolution (PIPE) algorithm, probabilistic representation of the program trees is used. Probabilities of instruction in each node in a maximal possible tree are used to model promising programs and generate new programs. Unused portions of the tree are simply cut before the evaluation of the program by a fitness function. Initially, the model is set so that the trees are generated at random. From the current population of programs ones that perform best are selected. These are then used to update the probabilistic model. The process is repeated until the termination criteria are met. PIPE is extension of EDA to genetic programming field.

In recent research, more improved approach is developed to capture multivariate distribution. Larrañaga et al.[7] considered the joint probability function of the continuous variable as a multivariate normal distribution. They used Gaussian networks to capture the continuous distribution, and proposed the $UMDA_c^G$, $MIMIC_c^G$, $EGNA_{ee}$, and $EGNA_{BGe}$. $UMDA_c^G$ is a continuous version of the *UMDA* (univariate marginal distribution algorithm), and the factorization of the joint density function is $f_l(x;\theta^l) = \prod_{i=1}^{n} f_l(x_i, \theta_t^l)$. Similarly $MIMIC_c^G$ is a continuous domain modification of *MIMIC* (mutual information maximizing input clustering) . To choose the permutation, they minimized the Kullback-Leibler information divergence between true density distribution and captured density function. $EGNA_{ee}$ is an algorithm based on the learning and simulation of Gaussian networks via detection of conditional independencies by edge exclusion tests. First a Gaussian network structure is learned, then the maximum likelihood estimates for the parameters of the learnt Gaussian network structure, and lastly the joint probability distribution function encoded by the Gaussian network is simulated. $EGNA_{BGe}$ is the same as $EGNA_{ee}$, but $EGNA_{BGe}$ uses BGe as the scoring metric.

Recently, a unified framework of EDA, IDEA, has been proposed[2]. The *Iterated Density Estimation Evolutionary Algorithm* (IDEA) is to be regarded separately from the algorithms that can be modelled by it, so IDEA framework is defined formally in both the case of discrete as well as continuous random variables. In IDEA framework, the Kullback-Leibler divergence with various distributions such as uniform, normal, or other distribution are used.

The FDA (Factorized Distribution Algorithm)[10] is also a good theoretical framework of EDA for continuous space as well as combinatorial optimization problem.

1. (Initialize) $X^0 \leftarrow$ generate M search points $\mathbf{x}_i^0$ from the prior distribution $P_0(\mathbf{x})$. Set generation count $t \leftarrow 0$.
2. (Likelihood) Estimate the likelihood s $p(D|A)$ of individual by evaluating their law fitness on data D.
3. (P-step) Update the parameter θ^* of a Helmholtz machine that maximizes $P(X^t|\theta)$.
4. (V-step) Generate L variations $X' = \{\mathbf{x}'_1, ..., \mathbf{x}'_L\}$ by sampling from the posterior predictive distribution $P_{t+1}(\mathbf{x}) = P(\mathbf{x}|X^t)$ using θ^* of the Helmholtz machine.
5. (S-step) Select M 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) Set $t \leftarrow t + 1$ and go to Step 2.

Figure 1: Outline of the Bayesian evolutionary algorithm using the Helmholtz machine for density estimation.

3 Bayesian Evolutionary Optimization

The Bayesian evolutionary algorithm (BEA) is a probabilistic model of evolutionary computation that is based on the Bayesian inductive principle [20, 21, 22]. While evolutionary algorithms iteratively produce the next generation of fitter individuals[1] using crossover and mutation operators, starting from an initial population of individuals, BEA do not use crossover or mutation operators explicitly. By defining the fittest model as the most probable model with respect to the data and the prior knowledge, the Bayes theorem can be used to estimate the posterior fitness of individuals from the prior fitness. The posterior probabilities are then used to generate plausible offspring individuals by sampling from the transition distribution formed by variation operators.

The outline of BEA using the Helmholtz machine for optimization problem is shown in Fig. 1. In essence, the BEA consists of three steps: 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. 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 are described in the next section. In the S-step, M best individuals are chosen into the next population X^{t+1} from the union of X^t and X'. More detailed explanation of BEA procedure will be followed:

Initially, a population X^0 of M individuals are generated from a prior distribution $P_0(\mathbf{x})$ such as a uniform distribution $(-R, R)$. R has a range of a given function. Then, the fitness value of the individual is observed and its likelihood $P(X^t|\theta)$ is computed, where θ is the parameter vector for the probability model, in this case the weights of Helmholtz machine.

Combining the prior and likelihood, we can compute the posterior probability $P(\theta|X^t)$ of individuals, using the Bayes rule:

$$P(\theta|X^t) = \frac{P(X^t|\theta)P(\theta)}{P(X^t)}. \quad (1)$$

Since $P(X^t)$ does not depend on the parameter vector θ, maximization of Eqn. (1) is equivalent to maximizing the numerator, i.e.

$$P(\theta|X^t) \propto P(X^t|\theta)P(\theta). \quad (2)$$

Note that, under the uniform prior for θ, maximization of Eqn. (2) is reduced to the problem of finding the maximum likelihood estimate θ^*:

$$\theta^* = \arg\max_\theta P(\theta|X^t) = \arg\max_\theta P(X^t|\theta). \quad (3)$$

We make use of this assumption and present a Bayesian evolutionary algorithm that performs optimization using a Helmholtz machine to estimate $P(X^t|\theta^*)$[23]. $P(\theta)$, the weights of Helmholtz machine is estimated by wake-sleep algorithm similar to the Estimation-Maximization method. The same Helmholtz machine is used through generations.

Offsprings are then sampled from the posterior distribution by sleep phase of wake-sleep algorithm. And offsprings are selected into the next generation by tournament selection. In the experiments, we use $L = 2M$. This is similar to the $(\mu + \lambda)$ evolution strategy with $\mu = M$, $\lambda = 2M$.

Note the similarity between the general structure of the BEA and the conceptual EDA [11]. The original BEA is more general than this[20, 22]; The one above is a EDA-like variant of it. More general BEAs calculate the maximum a posteriori probability rather than the maximum likelihood and the sample size increases as generation goes on.

4 Helmholtz Machine for Density Estimation

The Helmholtz machine is a connectionist system with multiple layers of neuron-like binary stochastic processing units connected hierarchically by two sets of weights, recognition weights and generative weights [3].

Bottom-up connections $\mathbf{R}$, shown as dashed lines in Fig. 2, implement the recognition model. This model is to infer a probability distribution over the underlying causes $\mathbf{y}$ (latent variables) of the input vector $\mathbf{x}$:

$$P(\mathbf{y}|\mathbf{x}, \mathbf{R}). \quad (4)$$

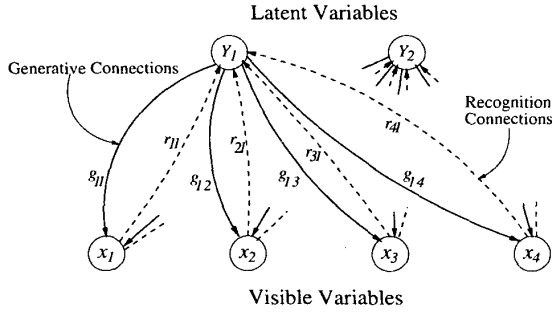

Figure 2: The Helmholtz machine (two-layer network).

Top-down connections $\mathbf{G}$, shown as solid lines in Fig. 2, implement the generative model. This second model is to reconstruct an approximation to the original input vector $\mathbf{x}$

$$P(\mathbf{x}|\mathbf{y}, \mathbf{G}) \quad (5)$$

from the underlying representation $\mathbf{y}$ captured by the hidden layer of the network. This enables to operate in a self-supervised manner. Both the recognition and generative models operate in a strictly feedforward fashion with no feedback.

In effect, Helmholtz machine estimates the distribution of the data points X^t, i.e. find the parameters $\theta^* = (\mathbf{R}^*, \mathbf{G}^*)$ that maximize the likelihood $P(X^t)$. After the distribution is learned, the samples from this distribution can be generated by randomly setting the latent variables and then propagating the values down to the input layer, just as the process in the sleep mode of the wake-sleep algorithm. This process is equivalent to sampling L offspring from the posterior predictive distribution since the following holds:

$$\begin{aligned} P_{t+1}(\mathbf{x}) &\equiv P(\mathbf{x}|X^t) & (6)\\ &= \int_Y \int_\Theta P(\mathbf{x}|\mathbf{y}, \theta) P(\mathbf{y}, \theta|X^t) d\theta d\mathbf{y} & (7)\\ &\approx \int_Y P(\mathbf{x}|\mathbf{y}, \theta^*) P(\mathbf{y}, \theta^*|X^t) d\mathbf{y} & (8)\\ &\approx \sum_{k=1}^{L} P(\mathbf{x}|\mathbf{y}_k, \theta^*), & (9) \end{aligned}$$

where $\theta^* = (\mathbf{R}^*, \mathbf{G}^*)$ is the maximum likelihood estimator for data X^t, and $P(\mathbf{x}|\mathbf{y}_k, \theta^*)$ is the generative model for the latent vectors $\mathbf{y}_k$ which are independently sampled from the uniform distribution.

We modified the Helmholtz machine to capture continuous distribution. Original Helmholtz machine use

$$p(\mathbf{y}) = \sigma\Big(\sum_i \mathbf{R}\mathbf{x}\Big), \quad (10)$$

where $\sigma(x) = 1/(1 + exp(-x))$ is the conventional sigmoid function, and decide the 0 or 1 probabilistically. Similarly our modified Helmholtz machine use sigmoid function, but to cover the range of given date set, we vary the slope parameter a of the sigmoid function; $\sigma(x) = 1/(1 + a \cdot exp(-x))$.

And instead of changing the value to 0 or 1, we use sigmoid function values. Each weight has its own normal distribution $N(b, \tau)$ as a noise to give a randomness. While original Helmholtz machine hold the probability in a node, our Helmholtz machine have the probability in a weight. More detailed explanation will be followed:

Base on the wake-sleep algorithm that proposed by Hinton et al. [4], we improved the wake-sleep algorithm to take a modification of Helmholtz machine into account. There are two phases in the algorithm: a *wake* phase and a *sleep* phase. In the *wake* phase, The units are driven bottom-up using the recognition weights, producing a representation of the input vector in the hidden layer. Therefore, the representation $\mathbf{y}_c$ produced in the hidden layer of the network provides a representation of the input vector $\mathbf{x}_c$:

$$\mathbf{y}_c = \mathbf{x}_c\mathbf{R} + \boldsymbol{\nu}_R. \tag{11}$$

where, $\boldsymbol{\nu}$ is a recognition noise vector which distribution is gaussian distribution with $N(\mathbf{b}_R, \boldsymbol{\tau}_R^2)$. Due to $\boldsymbol{\nu}_R$, $\mathbf{y}_c$ is set probabilistically. Although the nodes are driven by the recognition weights, only the generative weights are actually learned during the wake phase using locally available information and the simple delta rule [12]:

$$\mathbf{G}' = \mathbf{G} + \eta(\mathbf{x}_c - \mathbf{G}\mathbf{y}_c)\mathbf{y}_c, \tag{12}$$

where $\mathbf{G}$ is the generative weight vector, $\mathbf{x}_c$ is the c-th sample, $\mathbf{y}_c$ is the value of the latent variables, and η is the learning rate. The variance $\boldsymbol{\tau}_G^2$ of noise $\boldsymbol{\nu}_G$ is decayed by a constant α:

$$\boldsymbol{\tau}'^2_G = \alpha\boldsymbol{\tau}_G^2. \tag{13}$$

The generative bias of $\mathbf{b}_G$ is also updated by delta rule:

$$\mathbf{b}'_G = \mathbf{b}_G + \eta(\mathbf{x}_c - \mathbf{G}\mathbf{y}_c). \tag{14}$$

In effect, this phase of the learning process makes generative weights be adapted to increase the probability that they would reconstruct the correct activity vector in the layer below.

In the *sleep* phase of the algorithm, the recognition weights $\mathbf{R}$ are turned off. And all of the units in the network is driven using the generative weights, starting at the hidden layer and working down to the input units. Because the nodes are stochastic and the values of the hidden units, $\mathbf{y}$, are randomly chosen, repeating this process would typically give rise to many different *"fantasy"* vectors $\mathbf{x}_k$ on the input layer:

$$\mathbf{x}_k = \mathbf{y}_k\mathbf{G} + \boldsymbol{\nu}_G. \tag{15}$$

where, $\boldsymbol{\nu}_G$ is a generative noise vector which distribution is gaussian distribution with $N(\mathbf{b}_G, \boldsymbol{\tau}_G^2)$. $\boldsymbol{\nu}_G$ give randomness to x_k. These fantasies supply an unbiased sample of the network's generative model of the data. Having produced a fantasy, the recognition weights are adjusted by the simple delta rule [12]:

$$\mathbf{R}' = \mathbf{R} + \eta(\mathbf{y}_k - \mathbf{R}\mathbf{x}_k)\mathbf{x}_k, \tag{16}$$

where $\mathbf{R}$ is the recognition weight vector, $\mathbf{y}_k$ is the k-th latent vector, and $\mathbf{x}_k$ is the k-th fantasy vector. The recognition noise is decayed by α which is the same as the generative noise, Eqn. (13). And recognition bias is updated by

$$\mathbf{b}_R = \mathbf{b}_R + \eta(\mathbf{y}_k - \mathbf{R}\mathbf{x}_k). \tag{17}$$

The *sleep* phase uses only locally available information without reference to any observation. This is why offspring in the Bayesian evolutionary algorithm can be efficiently sampled from the distribution.

By the way, one of difficult problems is how to decide the structure of Helmholtz machine. Small network cannot learn the distribution well, but too large size will lead to overfitting and take a lot of time for learning. To solve this problem, we use a constructive algorithm for structure learning in Helmholtz machine[5]. A constructive algorithm starts with a small network and then adds additional hidden units and weights until a satisfactory solution is found, so it always searches for small networks to reduce the training time. But, since it is hard to decide when to stop the addition of hidden units, we use a greedy approach to network construction. After adding a new hidden unit with small random initial values, we train the *whole* network by wake-sleep algorithm, not only the new hidden unit.

5 Experimental Results

We have compared the results of Bayesian evolutionary algorithm (BEA) with Helmholtz machine with those of simple real-coded genetic algorithm (GA) on some functions such as Sphere function, Ackley function, Griewank function, Rastrigin function, Rosenbrock function, and test functions from Larrañaga et al. [8].

- Sphere function:

$$f_{sphere}(\mathbf{x}) = \sum_{i=0}^{N-1} x_i^2 \tag{18}$$

$where, \vec{x^*}=(0,\ldots,0)^N;\ f^*_{sphere}=0; -20 \le x_i \le 20$

- Ackley function:

$$f_{Ackley}(\mathbf{x}) = -20 \cdot exp\left(-0.2\sqrt{\frac{1}{N}\sum_{i=0}^{N-1} x_i^2}\right) - exp\left(\frac{1}{N}\sum_{i=0}^{N-1}\cos(2\pi\cdots x_i)\right) + 20 + e \tag{19}$$

$where, \vec{x^*}=(0,\ldots,0)^N;\ f^*_{Ackley}=0; -20 \le x_i \le 20$

- Rastrigin function:

$$f_{Rastrigin}(\mathbf{x}) = 10N + \sum_{i=0}^{N-1}\left(x_i^2 - 10\cos(2\pi x_i)\right) \tag{20}$$

$where, \vec{x^*}=(0,\ldots,0)^N;\ f^*_{Rastrigin}=0; -5.12 \le x_i \le 5.12$

Table 1: Mean fitness values with standard deviations and number of evaluations for Sphere function, Ackley function, Griewank function, Rastrigin function, and Rosenbrock function

		GA		BEA	
	N	Mean ± Stdev	# Evaluations	Mean ± Stdev	# Evaluations
Sphere	50	17.43 ± 1.17	19,930,400	$2.31 \times 10^{-5} \pm 2.28 \times 10^{-5}$	15,700
	100	20.37 ± 1.00	19,974,800	$1.75 \times 10^{-5} \pm 1.92^{-5}$	20,200
Ackley	50	1863.00 ± 132.02	19,919,800	$1.48 \times 10^{-3} \pm 7.61 \times 10^{-4}$	16,400
	100	4549.99 ± 280.06	19,905,000	$8.61 \times 10^{-4} \pm 4.90 \times 10^{-4}$	21,800
Griewank	50	18.08 ± 0.60	19,931,000	$1.42 \times 10^{-6} \pm 2.44 \times 10^{-6}$	28,100
	100	20.81 ± 0.90	19,978,800	$2.70 \times 10^{-6} \pm 6.72 \times 10^{-6}$	28,800
Rastrigin	50	221.72 ± 21.57	19,459,800	$2.32 \times 10^{-3} \pm 2.65 \times 10^{-3}$	17,800
	100	447.28 ± 22.70	19,792,800	$3.73 \times 10^{-4} \pm 7.33 \times 10^{-4}$	30,400
Rosenbrock	50	3941.472 ± 431.93	19,459,800	49.23 ± 0.47	13,100
	100	8909.16 ± 838.65	18,958,600	98.46 ± 0.41	20,200

- Griewank function:

$$f_{Griewank}(\mathbf{x}) = \frac{1}{4000 \cdot N} \cdot \sum_{i=0}^{N-1} x^2 - \prod_{i=0}^{N-1} \cos\left(\frac{x_i}{\sqrt{(i+1)}}\right) + 1 \quad (21)$$

$where, \vec{x^*} = (0,\ldots,0)^N;\ f^*_{Griewank} = 0; -600 \le x_i \le 600$

- Rosenbrock function:

$$f_{Rosenbrock}(\mathbf{x}) = \sum_{i=1}^{N} \left(100 \cdot (x_i - x_{i-1}^2)^2 + (1 - x_{i-1})^2\right) \quad (22)$$

$where, \vec{x^*} = (1,\ldots,1)^N; f^*_{Rosenbrock} = 0; -2.048 \le x_i \le 2.048$

- Test function 1:

$$f_{Test1}(\mathbf{x}) = \left\{10^{-5} + \sum_{i=1}^{d} |y_i|\right\}^{-1}$$
$$y_1 = x_1;\ y_i = y_{i-1} + x_i, i = 2, \ldots, d \quad (23)$$

$where, \vec{x^*} = (0,\ldots,0)^N;\ f^*_{Test1} = 10^5 - 0.16 \le x_i \le 0.16$

- Test function 2:

$$f_{Test2}(\mathbf{x}) = \sum_{i=1}^{d} \left((x_1 - x_i^2)^2 + (x_i - 1)^2\right) \quad (24)$$

$where, \vec{x^*} = (1,\ldots,1)^N; f^*_{Test2} = 0; -10 \le x_i \le 10$

- Test function 3:

$$f_{Test3}(\mathbf{x}) = 1 + \sum_{i=1}^{d} \frac{x_i^2}{4000} - \prod_{i=1}^{d} \cos\left(\frac{x_i}{\sqrt{i}}\right) \quad (25)$$

$where, \vec{x^*} = (0,\ldots,0)^N;\ f^*_{Test3} = 0; -600 \le x_i \le 600$

All functions except test function 1 have to minimize and have their minimum at 0. On the contrary, test function 1 is maximization function and has its maximum at 10^5. Besides, while $\mathbf{x} = (0,\ldots,0)$ have their optimum for most functions, $\mathbf{x} = (1,\ldots,1)$ has their optimum for Rosenbrock function and test function 2.

The GA used is the usual implementation that is based on one cut-point crossover, one point mutation, and roulette-wheel selection with elitist strategy. The parameters for GA were: maximum generation = 10,000, population size = 2,000, crossover rate = 0.9, and mutation rate = 0.01. The parameters of the BEA with the Helmholtz machine were: maximum generation = 50, population size = 1,000, learning rate = 0.001, noise decay rate = 0.999, and the number of learning iterations = 1,000. In each generation, new 1,000 offsprings were sampled. Tournament selection with tournament size two was used to select the next population, and also elitist strategy was used. For the first five bench mark functions, Helmholtz machine with two layer is used, otherwise for the last three test functions, Helmholtz machine with three layer is used to capture more complicate relationship. For constructive learning of network structure, initial Helmholtz machine has only one hidden unit in the hidden layer. In the case of three layered Helmholtz machine, the latent variables in the second hidden layer is set to one, and only latent variable of first hidden layer is added. And if there is no improvement during the first five generations, one hidden unit is added to the network. For objective comparisons, the parameter values for both methods were set as similarly as possible.

Tables 1 summarizes the experimental results of first five benchmark functions. In the table, N is the dimension of given function, 'Mean' is mean value of optimal over 20 runs, 'Stdev' stands for standard deviation of the optimal values, and '# Evaluations' is the average of number of evaluations to the optimal over 20 runs.

As shown in Table 1, Bayesian evolutionary algorithm using Helmholtz machine significantly outperforms the real-coded genetic algorithm in all functions. For the first four test functions, BEA can find quite good solutions using only one

Table 2: Mean values averaged on 100 runs for the test function 1, 2, and 3. Input dimension is 10 for all functions

	$UMDA_c^G$	$MIMIC_c^G$	$EGNA_{ee}$	$EGNA_{BGe}$	ES	BEA
Test function 1	53460	58775	100000	100000	5910	76646.08
Test function 2	0.13754	0.13397	0.09914	0.0250	0	1.64×10^{-8}
Test function 3	0.011076	0.007794	0.008175	0.012605	0.034477	7.90×10^{-9}

hidden units. We speculate that the good performance is affected by overall trend of four functions such as sphere function, Ackley function, Griewank function, and Rastrigin function. Even if there are many local optima around the global optima, overall shape of the functions is similar with that of sphere function. This is why Helmholtz machine with only one hidden unit can find good solutions. Even if there are higher order correlations in the function such as Rosenbrock function, BEA can find good solutions. Table 1 also describes that the standard deviations of BEA is quite large. It means that BEA with Helmholtz machine is unstable and sometimes converges to local optima. This unstableness is caused by wake-sleep algorithm for Helmholtz machine. There has been no general theoretical proof of the convergence of wake-sleep algorithm. This is a weak point of Helmholtz machine. But the empirical results are good enough to compensate this weak point. But one strange point is the results of the lower dimension cases ($N = 50$) are not better than the results of the higher dimension cases ($N = 100$) except for Rosenbrock function. To explain this behavior, we tested sphere function and Ackley function with dimension 150 and 200; besides test function 2 with dimension 30. For sphere function, we get the results 3.97×10^{-6} with $N = 150$ and 9.44×10^{-7} with $N = 200$; for Ackley function, 5.09×10^{-4} with $N = 150$ and 9.41×10^{-4} with $N = 200$; for test function 2, 4.69×10^{-7} with $N = 30$. Although the results go better as dimension increases in sphere function, the results of Ackley function are similar for the last three high dimensions and the results of test function 2 go worse. We speculate that this behavior is caused by characteristic of Helmholtz machine and benchmark functions.

The results of test functions from Larrañaga et al. [8] are explained in Table 2. As described in Section 2, $UMDA_c^G$ is a continuous version of the *UMDA*; $MIMIC_c^G$ is a continuous modification of *MIMIC* with Kullback-Leibler divergence; $EGNA_{ee}$ is an algorithm based on the learning and simulation of Gaussian networks using edge exclusion tests; $EGNA_{BGe}$ uses BGe scoring metric. 100 experiments for each function and algorithm were carried out. As shown in the Table 2, proposed methods show highly competitive results. Although BEA are slightly worse than EGNA for test function 1, they significantly outperform other functions.

Fig. 3 - Fig. 7 compares the fitness of BEA and GA with respect to the total number of fitness evaluations. The upper x axis stands for the number of fitness evaluations of BEA and the lower x axis represent GA. The average numbers of fitness evaluations to the best fitness are given in the Table 1. BEA can find good solutions faster than GA in terms of the

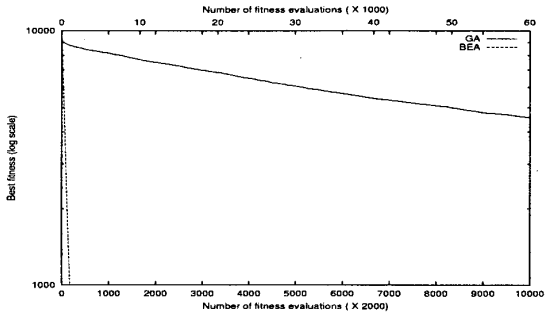

Figure 3: The number of fitness evaluations for sphere function with dimension $N = 100$.

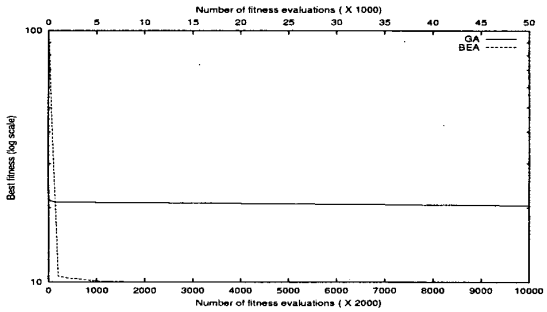

Figure 4: The number of fitness evaluations for Ackley function with dimension $N = 100$.

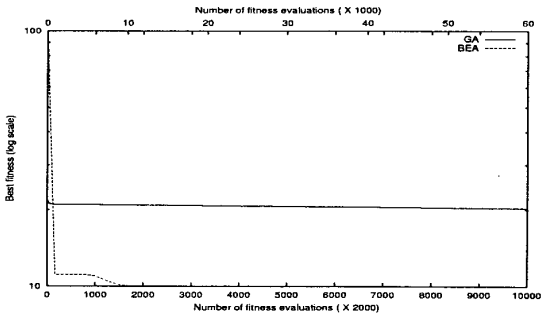

Figure 5: The number of fitness evaluations for Griewank function with dimension $N = 100$.

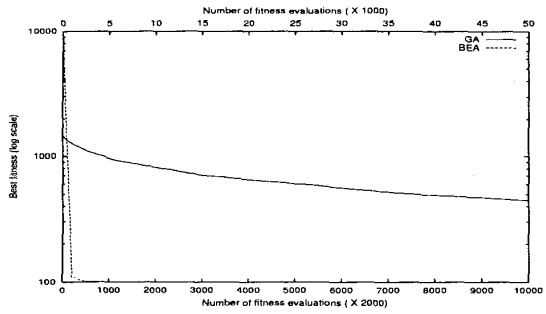

Figure 6: The number of fitness evaluations for Rastrigin function with dimension $N = 100$.

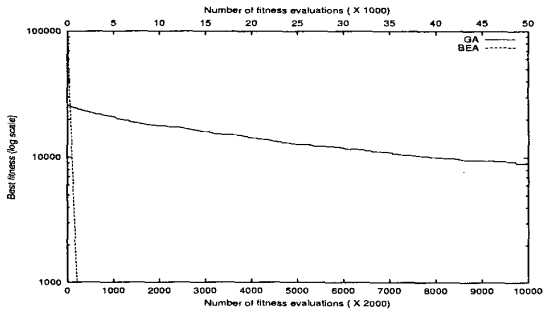

Figure 7: The number of fitness evaluations for Rosenbrock function with dimension $N = 100$.

number of fitness evaluations. Also BEA finds good solutions in the early generations. Even if it takes time for training of Helmholtz machine, BEA can find better solutions than GA using same execution time.

6 Conclusions

We present the Bayesian evolutionary algorithm (BEA) using Helmholtz machine as a continuous function optimization method. In the BEA framework, we modified Helmholtz machine to estimate the continuous distribution. And we test BEA for eight benchmark functions such as sphere function, Ackley function, Griewank function, Rastrigin function, Rosenbrock function, and three test functions from recent research. Our empirical results show that the Bayesian evolutionary algorithms outperform the real-coded genetic algorithms and other continuous estimation of distribution algorithms. Based on previous work[23] and this work, we can say that BEA with Helmholtz machine is a competitive optimization method and Bayesian evolutionary algorithm is a general theoretical framework for the estimation of distribution algorithm.

Future works include the improvement of the wake-sleep algorithm and network construction algorithm to capture more reasonable distribution. And we want to compare the results with evolutionary strategy and another previous EDA methods.

Acknowledgments

This research was supported in part by the Korea Science and Engineering Foundation (KOSEF) under Grant 981-0920-107-2, by the Korea Ministry of Science and Technology through KISTEP under Grant BR-2-1-G-06, and by the Ministry of Education under the BK21-IT Program.

Bibliography

[1] T. Bäck, *Evolutionary Algorithms in Theory and Practice*. Oxford Univ. Press, 1996.

[2] P. A. N. Bosman and D. Thierens, "Expanding from Discrete to Continuous Estimation of Distribution Algorithms: The IDEA", it Parallel Problem Solving from Nature - PPSN VI, pp. 767-776, Springer-Verlag, 2000.

[3] P. Dayan, G. E. Hinton, R. M. Neal, and R. S. Zemel, "The Helmholtz machine", *Neural Computation*, **7**: 1022-1037, 1995.

[4] G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal, "The wake-sleep algorithm for unsupervised neural networks", *Science*, **268**: 1158-1160, 1995.

[5] T.-Y. Kwok and D.-Y. Yeung, "Constructive Algorithms for Structure Learning in Feedforward Neural Networks for Regression Problems", *IEEE Transactions on Neural Networks*, **8**(3): 630-645, IEEE, 1997.

[6] P. Larrañaga, R. Etxeberria, J. A. Lozano, B. Sierra, I. Inza, and J. M. Peña, "A review of the cooperation between evolutionary computation and probabilistic graphical models", *Proc. of the Second Symposium on Artificial Intelligence, CIMAF 99*, pp. 314-324, Adaptive Systems, 1999.

[7] P. Larrañga, R. Etxeberria, J. A. Lozano, and J. M. Peña, "Optimization by learning and simulation of Bayesian and Gaussian networks", *Technical Report EHU-KZAA-IK-4/99*, http://www.sc.edu.es/ccwbayes/postscript/kzaa-ik-04-99.ps, 1999.

[8] P. Larrañaga, R. Etxeberria, J. A. Lozano, and M. Peña, "Optimization in continuous domains by learning and simulation of Gaussian networks," *Proceedings of the 2000 Genetic and Evolutionary Computation Conference Workshop Program*, pp. 201-204, 2000.

[9] S. Margetts and A. J. Jones, "Phlegmatic Mappings for Function Optimization with Genetic Algorithms", *Proc. the Genetic and Evolutionary Computing Conference*, pp. 82-89, Morgan Kaufmann, 2000.

[10] H. Mühlenbein and T. Mahnig "FDA - A scalable evolutionary algorithm for the optimization of additively decomposed functions", *Evolutionary Computation*, **7**(4):353-376, 1999.

[11] H. Mühlenbein, T. Mahnig, and A. Ochoa, "Schemata, distributions and graphical models in evolutionary optimization", *Journal of Heuristics*, **5**:215-247, 1999.

[12] R. M. Neal and P. Dayan, "Factor analysis using delta-rule wake-sleep learning", *Neural Computation*, **9**:1781-1803, 1997.

[13] M. Pelikan, D. E. Goldberg, and F. Lobo, "A Survey of Optimization by Building and Using Probabilistic Models", *Computational Optimization and Applications*, Kluwer, 2000 (In printing).

[14] S. Rudlof and M. Köppen, "Stochastic hill climbing with learning by vectors of normal distributions", *The first Online Workshop on Soft Computing*, http://www.uchikawa.nuie.nagoya-u.ac.jp/wsc1/papers/p077.html, 1996.

[15] R. P. Salustowicz and J. Schmidhuber, "Probabilistic incremental program evolution: stochastic search through program space", *Machine Learning: ECML-97, Lecture Notes in Artificial Intelligence, Vol. 1224*, pp. 213-220, Springer-Verlag, 1997.

[16] M. Sebag and A. Ducoulombier, "Extending population-based incremental learning to continuous search spaces", *Parallel Problem Sovling from Nature - PPSN V*, pp. 418-427, Springer-Verlag, 1998.

[17] I. Servet, L. Trave-Massuyes, D. Stern, "Telephone network traffic overloading diagnosis and evolutionary computation techniques", *Proc. of the Third European Conf. on Artificial Evolution*, pp. 137-144, 1997.

[18] R. Thomsen, P. Rickers, and T. Krink, "A Religion-Based Spatial Model for Evolutionary Algorithm", *Parallel Problem Solving from Nature - PPSN VI*, pp. 817-826, Springer-Verlag, 2000.

[19] K. Weicker and N. Weicker, "On the improvement of coevolutionary optimizers by learning variable interdependencies", *Proc. 1999 Congress on Evolutionary Computation*, pp. 1627-1632, 2000.

[20] B.-T. Zhang, "A Bayesian framework for evolutionary computation". In *Proc. 1999 Congress on Evolutionary Computation* (CEC99), IEEE Press, pp. 722-727, 1999.

[21] B.-T. Zhang, "Bayesian methods for efficient genetic programming". *Genetic Programming and Evolvable Machines*, **1**(3):217-242, 2000.

[22] B.-T. Zhang, "Bayesian Evolutionary Algorithms for Learning and Optimization", *Proc. 2000 Congress on Evolutionary Computation Workshop*, pp. 220-222, 2000.

[23] B.-T. Zhang and S.-Y. Shin, "Bayesian Evolutionary Optimization Using Helmholtz Machines", *Parallel Problem Solving from Nature - PPSN VI*, pp. 827-836, Springer-Verlag, 2000.