# Bayesian Evolutionary Optimization Using Helmholtz Machines

Byoung-Tak Zhang and Soo-Yong Shin

Artificial Intelligence Lab. (SCAI)
School of Computer Science and Engineering
Seoul National University
Seoul 151-742, Korea
{btzhang, syshin}@scai.snu.ac.kr
http://scai.snu.ac.kr/

**Abstract.** Recently, several evolutionary algorithms have been proposed that build and use an explicit distribution model of the population to perform optimization. One of the main issues in this class of algorithms is how to estimate the distribution of selected samples. In this paper, we present a Bayesian evolutionary algorithm (BEA) that learns the sample distribution by a probabilistic graphical model known as Helmholtz machines. Due to the generative nature and availability of the wake-sleep learning algorithm, the Helmholtz machines provide an effective tool for modeling and sampling from the distribution of selected individuals. The proposed method has been applied to a suite of GA-deceptive functions. Experimental results show that the BEA with the Helmholtz machine outperforms the simple genetic algorithm.

## 1 Introduction

Evolutionary algorithms are typically used to solve function optimization problems, i.e. to find the point $\mathbf{x}^*$ that maximizes the objective function:

$$\mathbf{x}^* = \arg\max_{\mathbf{x}}\{f(\mathbf{x})\}. \tag{1}$$

Conventional evolutionary algorithms solve this problem by iteratively generating populations $X^t$ of search points $\mathbf{x}_i^t$ until $\mathbf{x}^*$ is found or the best solution $\mathbf{x}_{best}^t$ at generation $t$ is acceptable [3]. This class of algorithms does not use an explicit model of the sample population; They just generate new points based on old points. Therefore, it is difficult to capture the structure of the objective function.

An alternative way is to model the population explicitly using a probability density function. 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 the population of good solutions and use the constructed model to guide further search [2, 5, 9, 11]. These methods are generally known as the estimation

of distribution algorithms or EDAs [8]. 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.

One of the main issues in distribution-based optimization is how to build and sample from the distribution of the population. Several methods have been proposed, including the methods based on dependency chains [5], dependency trees [2], factorization [9], neural trees [13], Bayesian networks [8, 12], and genetic programs [14].

In this paper, we present a method that estimates the sample distribution using a graphical learning model known as Helmholtz machines. The method is implemented as a Bayesian evolutionary algorithm (BEA), a probabilistic model of evolutionary computation [13]. The Helmholtz machine is a multilayer network [4] and can find the hidden structure in a data set. Even if the structure has complex relationship, the Helmholtz machine can model the data dependency by a hierarchical network. The wake-sleep algorithm [6] provides a learning mechanism for capturing the underlying structure of the data. In addition, since the Helmholtz machine is a generative model, generation of new samples from the model is efficient. Our experimental evidence supports that Helmholtz machines are effective for estimation and simulation of the population distribution.

The paper is organized as follows. In Section 2, previous work is briefly reviewed. Section 3 presents the Bayesian evolutionary algorithm using the Helmholtz machine. We also describe the architecture and learning algorithm of the Helmholtz machine. Section 4 reports the experimental results. Conclusions are drawn in Section 5.

## 2 Optimization by Distribution Estimation

A simplest way for distribution estimation is to consider each variable in a problem independently and generate new solutions by only preserving the proportions of the values of all variables independently of the remaining solutions. Population-based incremental learning (PBIL) 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 [7] 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. Both PBIL and UMDA are, however, not appropriate for learning higher-order dependency.

To capture more complex dependency, structures that can express higher-order statistics are necessary. De Bonet et al. [5] present a population-based algorithm using second-order statistics, called MIMIC (mutual information maximizing input clustering). It uses a chain structure to express conditional probabilities. Baluja and Davies [2] propose to use dependency trees to learn second-order

probability distributions and use these statistics to generate new solutions. Pelikan and Mühlenbein [11] suggest the bivariate marginal distribution algorithm (BMDA) as an extension of the UMDA. BMDA uses the pairwise dependencies in order to improve algorithms that use simple univariate marginal distribution. MIMIC, dependency trees, and BMDA can cover pairwise interactions.

Mühlenbein and Mahnig [9] present the factorized distribution algorithm (FDA). Here, the distribution is decomposed into various factors or conditional probabilities. Distribution factorization is obtained by analyzing the problem structure. FDA is able to cover interactions of higher-order and combine important partial solutions effectively. Even though it works very well on additively decomposable problems, FDA requires the prior information about the problem in the form of a problem decomposition and its factorization. Pelikan et al.[12] describe the Bayesian optimization algorithm (BOA) that uses techniques from the field of modeling data by Bayesian networks in order to estimate the joint distribution of promising solutions. BOA is also capable of expressing higher-order interactions. In principle, BOA does not require any predefined structure of a problem. But, if no information on the structure is given, BOA also has some difficulties since the construction of a Bayesian network is not an easy task without any prior information.

Zhang [13] presents Bayesian evolutionary algorithms (BEAs) where optimization is formulated as a probabilistic process of finding an individual with the maximum a posteriori probability (MAP). In previous work, tree-structured neural networks [15] and Lisp-like symbolic programs [14] were used to model the distribution of the data points. In the following section, we present another implementation of the BEA that uses a Helmholtz machine for the estimation of the density of search points and for generating new search points from the estimated density.

# 3 The Bayesian Evolutionary Algorithm Using Helmholtz Machines

## 3.1 The Bayesian Evolutionary Algorithm for Optimization

The Bayesian evolutionary algorithm is a probabilistic model of evolutionary computation that is based on the Bayesian inductive principle [13]. Initially, a population $X^0$ of $M$ individuals are generated from a prior distribution $P_0(\mathbf{x})$. Then, the fitness values of the individuals are observed and its likelihood $P(X^t|\theta)$ is computed, where $\theta$ is the parameter vector for the probability model. Combining the prior and likelihood, we can compute the posterior probability $P(\theta|X^t)$ of individuals, using the Bayes rule:

$$P(\theta|X^t) = \frac{P(X^t|\theta)P(\theta)}{P(X^t)}. \tag{2}$$

Since $P(X^t)$ does not depend on the parameter vector $\theta$, maximization of Eqn. (2) is equivalent to maximizing the numerator, i.e.

$$P(\theta|X^t) \propto P(X^t|\theta)P(\theta). \tag{3}$$

Offspring are then sampled from the posterior distribution and selected into the next generation.

Note that, under the uniform prior for $\theta$, maximization of Eqn. (3) is reduced to the problem of finding the maximum likelihood estimate $\theta^*$:

$$\theta^* = \arg\max_{\theta} P(\theta|X^t) = \arg\max_{\theta} P(X^t|\theta). \tag{4}$$

In this paper, we make use of this assumption and present a Bayesian evolutionary algorithm that performs optimization using a Helmholtz machine to estimate $P(X^t|\theta^*)$. The algorithm is summarized in Figure 1.

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. (P-step) Estimate the parameter $\theta^*$ of a Helmholtz machine that maximizes $P(X^t|\theta)$.
3. (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 $\theta^*$ of the Helmholtz machine.
4. (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 $f(\mathbf{x})$.
5. (Loop) Set $t \leftarrow t + 1$ and go tc Step 2.

**Fig. 1.** Outline of the Bayesian evolutionary algorithm using the Helmholtz machine for density estimation.

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. The same Helmholtz machine is used though the generations. 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 subsection. In the S-step, $M$ best individuals are chosen into the next population $X^{t+1}$ from the union of $X^t$ and $X'$. In the experiments, we use $L = 10M$. This is similar to the $(\mu + \lambda)$ evolution strategy [3] with $\mu = M$, $\lambda = 10M$.

Note the similarity between the general structure of the BEA and the conceptual EDA [8]. The original BEA [15] is more general than this; 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.

### 3.2 Distribution Estimation by the Helmholtz Machine

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 [4]. Bottom-up connections

**R**, shown as dashed lines in Fig. 2, implement the recognition model. This model is to infer a probability distribution over the underlying causes **y** (latent variables) of the input vector **x**:

$$P(\mathbf{y}|\mathbf{x}, \mathbf{R}). \tag{5}$$

Top-down connections **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 **x**

$$P(\mathbf{x}|\mathbf{y}, \mathbf{G}) \tag{6}$$

from the underlying representation **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.

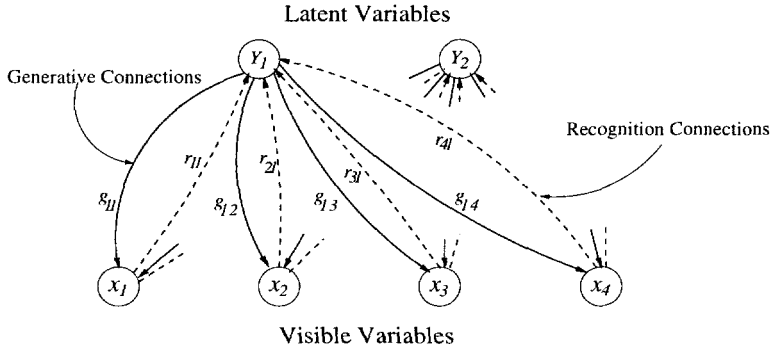


**Fig. 2.** The Helmholtz machine (two-layer network).

Hinton et al. [6] describe a stochastic algorithm, called the wake-sleep algorithm, to calculate the recognition and generative weights of the Helmholtz machine. 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 $\mathbf{x}_c$:

$$\mathbf{y}_c = h(\mathbf{x}_c, \mathbf{R}). \tag{7}$$

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 [10]:

$$\mathbf{G}' = \mathbf{G} + \eta(\mathbf{x}_c - \mathbf{G}\mathbf{y}_c)\mathbf{y}_c, \tag{8}$$

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 $\eta$ is the learning rate. 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 gives rise to many different *"fantasy"* vectors $\mathbf{x}$ on the input layer:

$$\mathbf{x}_k = g(\mathbf{y}_k, \mathbf{G}). \tag{9}$$

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 [10]:

$$\mathbf{R}' = \mathbf{R} + \eta(\mathbf{y}_k - \mathbf{R}\mathbf{x}_k)\mathbf{x}_k, \tag{10}$$

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 "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.

In effect, the 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:

$$P_{t+1}(\mathbf{x}) \equiv P(\mathbf{x}|X^t) = \int_Y \int_\Theta P(\mathbf{x}|\mathbf{y},\theta)P(\mathbf{y},\theta|X^t)d\theta d\mathbf{y} \tag{11}$$

$$\approx \int_Y P(\mathbf{x}|\mathbf{y},\theta^*)P(\mathbf{y},\theta^*|X^t)d\mathbf{y} \tag{12}$$

$$\approx \sum_{k=1}^{L} P(\mathbf{x}|\mathbf{y}_k,\theta^*), \tag{13}$$

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.

## 4 Experimental Results

Experiments have been performed on three benchmark problems from the literature [11]. They are the one-max function, quadratic function, and 3-deceptive function as defined below.

- One-max function:

$$f_{onemax}(\mathbf{x}) = \sum_{i=0}^{n-1} x_i, \tag{14}$$

where $x_i$ is the value on the $i$th position in bit string $\mathbf{x}$. The one-max function is a simple linear function that is just the sum of all bits in a string.

- Quadratic function:

$$f_{quadratic}(\mathbf{x}, \pi) = \sum_{i=0}^{\frac{n}{2}-1} f_2(\mathbf{x}_{\pi(2i)}, \mathbf{x}_{\pi(2i+1)}), \tag{15}$$

where $\pi$ is defined as

$$\pi_k(i) = \left\lfloor \frac{n(i \bmod k) + 1}{k} \right\rfloor. \tag{16}$$

For this problem, the permutation of order 2, $\pi_2$, was used and $f_2$ is defined as

$$f_2(u, v) = 0.9 - 0.9(u + v) + 1.9uv. \tag{17}$$

With both arguments equal to 1 we get $f_2(1, 1) = 1$. Therefore, the optimum is clearly in the string with all 1's.

- 3-deceptive function:

$$f_{3deceptive}(\mathbf{x}, \pi) = \sum_{i=0}^{\frac{n}{3}-1} f_3(\mathbf{x}_{\pi(3i)} + \mathbf{x}_{\pi(3i+1)} + \mathbf{x}_{\pi(3i+2)}), \tag{18}$$

where $\pi_3$ is used and $f_3$ is defined as

$$f_3(u) = \begin{cases} 0.9 & \text{if } u = 0, \\ 0.8 & \text{if } u = 1, \\ 0 & \text{if } u = 2, \\ 1 & \text{otherwise.} \end{cases} \tag{19}$$

The performance of the Helmholtz machine was compared with that of the simple genetic algorithm (sGA). The sGA we use is the usual implementation that is based on one cut-point crossover, one point mutation, and roulette-wheel selection. The parameters for sGA were: maximum generation = $10^6$, population size = $10^3$, crossover rate = 0.5, and mutation rate = 0.01. The parameters of the BEA with the Helmholtz machine were: maximum generation = $10^3$, population size = $10^3$, learning rate = 0.001, and the number of learning iterations = $10^3$.To select the next population, upper 10% truncation selection was used. For objective comparison, the parameter values for both methods were set as similar as possible. We use a 2-layer Helmholtz machine for the Beysian evolutionary algorithm, and only one latent variable was used for solving both the

**Table 1.** Results for the one-max function.

| Prob. Size | Succ % BEA | Succ % sGA | #iterations BEA | #iterations sGA | CPU time (second) BEA | CPU time (second) sGA |
|---|---|---|---|---|---|---|
| 50 | 100 | 100 | 5.0 | 116.8 | 295 | 6 |
| 100 | 100 | 100 | 27.8 | 625.7 | 2,758 | 30 |
| 150 | 100 | 100 | 54.2 | 1925.3 | 8,480 | 116 |
| 200 | 100 | 100 | 70.0 | 4576.0 | 13,912 | 324 |
| 250 | 100 | 100 | 96.6 | 10103.6 | 24,584 | 950 |
| 300 | 100 | 100 | 139.6 | 26549.7 | 42,549 | 2737 |

**Table 2.** Results for the quadratic function.

| Prob. Size | Succ % BEA | Succ % sGA | #iterations BEA | #iterations sGA | CPU time (second) BEA | CPU time (second) sGA |
|---|---|---|---|---|---|---|
| 50 | 100 | 100 | 20.1 | 367.2 | 1,073 | 27 |
| 100 | 100 | 100 | 48.0 | 18833.1 | 4,791 | 2,218 |
| 150 | 100 | 0 | 58.6 | - | 9,229 | - |
| 200 | 100 | 0 | 85.1 | - | 17,061 | - |
| 250 | 100 | 0 | 118.8 | - | 30,222 | - |
| 300 | 100 | 0 | 134.0 | - | 41,173 | - |

**Table 3.** Results for the 3-deceptive function.

| Prob. Size | Succ % BEA | Succ % sGA | #iterations BEA | #iterations sGA | CPU time (second) BEA | CPU time (second) sGA |
|---|---|---|---|---|---|---|
| 15 | 100 | 30 | 3.4 | 49500.1 | 92 | 2623 |
| 30 | 100 | 0 | 6.3 | - | 264 | - |
| 45 | 100 | 0 | 7.8 | - | 494 | - |
| 60 | 100 | 0 | 10.5 | - | 827 | - |
| 90 | 100 | 0 | 13.1 | - | 1587 | - |
| 120 | 100 | 0 | 15.7 | - | 2436 | - |

**Table 4.** Results for various pop size on the one-max function with problem size 100.

| Prob. Size | Succ % BEA | Succ % sGA | #iterations BEA | #iterations sGA |
|---|---|---|---|---|
| 50 | 100 | 0 | 61.5 | - |
| 100 | 100 | 0 | 59.0 | - |
| 500 | 100 | 100 | 48.7 | 1975.6 |
| 1000 | 100 | 100 | 46.7 | 625.7 |

one-max and quadratic functions. While more than one latent variable was used for 3-deceptive function. A constructive algorithm was devised to solve the for 3-deceptive function. It starts from one latent variable, and increases the number of latent variables during learning.

Tables 1 - 4 summarize the results. The results shown are average values over 10 runs. The CPU time was measured on Intel Pentium II - 350 Mhz PC with Windows 2000. The algorithm is considered as converged if an optimal solution is found. The entry of the tables marked with '-' means that the algorithm did not find the optimal solution. The results show that the Bayesian evolutionary algorithms with Helmholtz machines outperform the simple genetic algorithm both in the success rate and the number of iterations. BEAs find the optimal solutions all the time while the simple GA finds solutions for easy problems such as one-max and small size instances of the quadratic and 3-deceptive function. It can be observed that from Table 4 that the BEA with Helmholtz machine can solve the problem using a very small data set. Even though the population size is 50, BEA can found the optimal solution. The improved algorithms, BEA with constructive Helmholtz machines could solve the 3-deceptive function very efficiently, significantly outperforming the simple genetic algorithm.

The number of iterations for the sGA denotes the number of generations, while that for the BEA with the Helmholtz machine denotes the number of sampling the population, excluding the distribution estimation (learning) time. Therefore, it is hard to infer the CPU time from the number of iterations only. For comparing the real evaluation time, we measured the CPU time for BEA with Helmholtz machines and the simple genetic algorithm. Even if BEAs took more CPU time than sGAs, BEAs could solve larger size problems. It is also interesting that the number of evaluations for sGAs grows exponentially while that for BEAs grows almost linearly.

## 5 Conclusions

We presented a distribution estimation algorithm that is based on the Helmholtz machine. Our empirical results show that the Bayesian evolutionary optimization algorithms using Helmholtz machines outperform the simple genetic algorithms in several conditions. The superiority of the probabilistic algorithms tend to grow linearly as the problem complexity and size increase and solve the using very small data sets. Future work includes the analysis of the effect of the number of latent variables and the number of hidden layers in the Helmholtz machine for more effective estimation of population distribution.

## 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 BK21 program.

## References


1. Baluja, S. and Caruana, R., "Removing the genetics from the standard genetic algorithm", Technical Report CMU-CS-95-141, Carnegie Mellon University, 1995.
2. Baluja, S. and Davies, S., "Using optimal dependency-trees for combinatorial optimization: learning the structure of the search space", *Proc. 14th Int. Conf. on Machine Learning*, pp. 30-38, Morgan-Kaufmann, 1997.
3. Bäck, T., *Evolutionary Algorithms in Theory and Practice*. Oxford Univ. Press, 1996.
4. Dayan, P., Neal, G.E., and Zemel, R.S., "The Helmholtz machine", *Neural Computation*, **7**: 1022-1037, 1995.
5. De Bonet, J.S., Isbell, C.L., and Viola, P., "MIMIC: Finding optima by estimating probability densities", *NIPS 9*, pp. 424-430, The MIT Press, 1997.
6. Hinton, G.E., Dayan, P., Frey, B.J., Neal R. M., "The wake-sleep algorithm for unsupervised neural networks", *Science*, **268**: 1158-1160, 1995.
7. Mühlenbein, H. and Paaß, G., "From recombination of genes to the estimation of distributions I: Binary parameters", *PPSN IV*, LNCS 1141, pp. 178-187, Springer, 1996.
8. Mühlenbein, H., Mahnig, T., and A. Ochoa, "Schemata, distributions and graphical models in evolutionary optimization", *Journal of Heuristics*, **5**:215-247, 1999.
9. Mühlenbein, H. and Mahnig, T., "FDA - A scalable evolutionary algorithm for the optimization of additively decomposed functions", *Evolutionary Computation*, **7**(4):353-376, 1999.
10. Neal, R.M. and Dayan, P., "Factor analysis using delta-rule wake-sleep learning", *Neural Computation*, **9**:1781-1803, 1997.
11. Pelikan, M. and Mühlenbein, H., "The bivariate marginal distribution algorithm", *Advances in Soft Computing - Engineering Design and Manufacturing*, pp. 521-535, London: Springer-Verlag, 1999.
12. Pelikan, M., Goldberg, D.E., and Cantú-Paz, E., "BOA: The Bayesian optimization algorithm", *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 525-532, CA: Morgan Kaufmann, 1999.
13. Zhang, B.-T., A Bayesian framework for evolutionary computation. In *Proc. 1999 Congress on Evolutionary Computation* (CEC99), IEEE Press, pp. 722-727, 1999.
14. Zhang, B.-T., Bayesian methods for efficient genetic programming. *Genetic Programming and Evolvable Machines*, **1**(3):217-242, 2000.
15. Zhang, B.-T. and Joung, J.-G., Efficient model induction by a Bayesian evolutionary algorithm with incremental data inheritance, Technical Report SCAI-98-017, Artificial Intelligence Lab (SCAI), Seoul National University, August 1998.