Estimation of Distribution Programming: EDA-based Approach to Program Generation

Kohsuke Yanai and Hitoshi Iba

Dept. of Frontier Informatics, Graduate School of Frontier Sciences,
The University of Tokyo,
5-1-5 Kashiwanoha, Kashiwa-shi, Chiba 277-8561, Japan
{yanai,iba}@iba.k.u-tokyo.ac.jp

Summary. We describe a framework for program evolution with an EDA-based approach. In this framework, the probability distribution of programs is estimated using a Bayesian network, and individuals are generated based on the estimated distribution. Considering that a dependency relationship of nodes in a program tree is explicit, i.e. the dependency relationship is strong between a parent node and its child node in a program expressed as a tree structure, we have chosen a Bayesian network as the distribution model of programs.

In order to demonstrate the effectiveness of our approach, this chapter shows results of comparative experiments with Genetic Programming. Thereafter, we discuss how Estimation of Distribution Programming works and the transitions of the evolved programs that are the forte of our methods. We also analyze the performance of a hybrid system which combines Estimation of Distribution Programming and Genetic Programming.

1 Introduction

In this chapter, we describe a program evolution method based on a probabilistic model and investigate the behavior of the proposed system.

A well-known technique for a program search is Genetic Programming (GP) [9]. Although various types of crossover and mutation operators were proposed for GP[1] there have been very few basic algorithms comparable to GP. We use a program evolution method which has different mechanisms from GP, and show that some of the GP difficulties can be solved effectively[2].

This chapter proposes Estimation of Distribution Programming (EDP) based on a probability distribution expression using a Bayesian network. EDP is a search

[1] For example, uniform crossover and one-point crossover [16], homologous crossover and size fair crossover [10], depth-dependent crossover [8] [7], macromutation [2], self-adaptive crossover [1], recombinative guidance crossover [6], and so on.

[2] It is well known that GP search space is significantly constrained [5], and that the bloat control is difficult [11]. Other GP difficulties have been reported in solving a royal tree problem [18] and a max problem [17].

method that uses an EDA-like approach to solve GP-applicable problems. In EDA, it is important to assume a gene locus dependency relationship. In a program tree this relationship is strong between the parent node and its child node, so that it is expected that the EDA approach will be effective for solving tree structure search problems [21]. We compare the performance of EDP and GP on several benchmark tests, and discuss the trends of problems that are the forte of EDP.

We also discuss the performance of a hybrid system which consists of EDP and GP. Applying the hybrid system of EDP and GP to a function regression problem, we discover some important tendencies in the behavior of this hybrid system. The hybrid system is not only superior to pure GP in a search performance but also have interesting features in program evolution. More tests reveal how and when EDP and GP compensate for each other.

2 Estimation of Distribution Programming

2.1 Algorithm of EDP

We give an outline of the proposed algorithm. EDP starts with a randomly generated population. Secondly, each individual in the current population is evaluated by a fitness function and assigned its fitness value. Next, superior individuals with high fitness values are selected, and a new distribution is estimated based on those selected individuals (see Sect. 2.3). We use the elitist strategy and then individuals are generated by using a newly acquired distribution (see Sect. 2.4). The estimation of distribution and the program generation are repeated until a termination criterion is met. Figure 1 indicates a pseudo code of EDP.

Initial Population

According to function node generation probability P_F and terminal node generation probability P_T $(1 - P_F)$, initial M individuals are generated randomly, where M is

Let P be a population, S a set of selected individuals, D a distribution, E_S an elite size, and M a population size.

```
1. $P$ := Generate_Programs_Randomly
2. While (True)
3.     Evaluate_Individuals($P$)
4.     If (termination_criterion) Return($P$)
5.     $S$ := Selection($P$)
6.     $D$ := Estimate_Distribution($S$)
7.     $P$ := Elite_Selection($P$, $E_S$)
8.     $P$ := P + Generate_Individuals($D$, $M - E_S$)
```

Fig. 1. Pseudo code of EDP

the population size. However, if tree size limitation is reached, terminal nodes are generated. Let F be the function node set and let T be the terminal node set. For example, the probabilities of function node "$+$" and terminal node "x" are given:

If tree size limitation is not reached,

$$\begin{cases} P(X = \text{“}+\text{”}) & = P_F \times \frac{1}{|F|} \\ P(X = \text{“}x\text{”}) & = P_T \times \frac{1}{|T|} \end{cases} \tag{1}$$

If tree size limitation is reached,

$$\begin{cases} P(X = \text{“}+\text{”}) & = 0 \\ P(X = \text{“}x\text{”}) & = \frac{1}{|T|} \end{cases} \tag{2}$$

EDP Operator

Superior individuals with high fitness values are selected within sampling size S_S, and a new distribution is estimated based on those selected individuals. We use the elitist strategy, i.e. elite E_S individuals are selected from the population in the order of fitness superiority and copied to the new population, where E_S is the elite size.

Then the remaining population, that is $M - E_S$ individuals, is generated by using a newly acquired distribution. This new distribution is considered better than the previous one because it samples superior individuals in the population.

2.2 Distribution Model

We use a Bayesian network as the distribution model of programs. Values of probabilistic variables are symbols for each node in the program tree. Assign the index numbers to each node of evolving programs as in Fig. 2, the range of probabilistic variable X_i is the symbols of node i, that is, $X_i \in T \cup F$.

For instance, assume $F = \{+, -, *, /\}$ and $T = \{x_1, x_2\}$,

$$P(X_5 = \text{“}+\text{”} | X_2 = \text{“}/\text{”}) = \frac{2}{7} \tag{3}$$

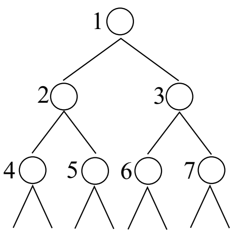

Fig. 2. Program tree

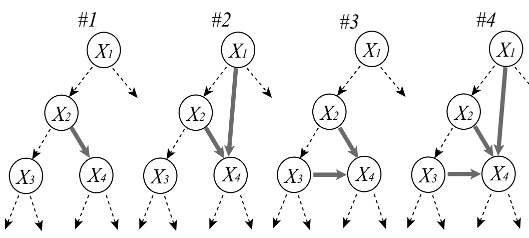

Fig. 3. Efficient network topology

means that the conditional probability that node 5 becomes " $+$ " is $\frac{2}{7}$ if node 2 is "$/$". C_i is the set of probabilistic variables which X_i is dependent on. In the former example, $C_5 = \{X_2\}$.

The topology of a Bayesian network is fixed during evolution, and only conditional probability tables are learned by sampling superior individuals in a population. Let d_{max} be the depth of a Bayesian network. We assume that the max arity of node symbols is 2 in this chapter. Although EDP cannot generate a larger program than a complete binary tree with a depth d_{max}, it can generate a smaller one.

There are several efficient topologies of a Bayesian network as indicated in Fig. 3. The simplest one, that is, $\#1$ in Fig. 3, is used for our experiments. The topology of a Bayesian network is tree-like and it is the same as program's topology. In this model, the probability of each node in a program tree is dependent on only its parent node symbol. This is based on the assumption that a dependency relationship is strong between the parent node and its child nodes.

2.3 Estimation of Distribution

The probability distribution is updated incrementally [3] as follows:

$$P_{t+1}(X_i = x|C_i = c) = (1-\eta)\hat{P}(X_i = x|C_i = c) + \eta P_t(X_i = x|C_i = c) \quad (4)$$

where $P_t(X_i = x|C_i = c)$ is the distribution of the $t\mathrm{th}$ generation and $\hat{P}(X_i = x|C_i = c)$ is the distribution estimated based on superior individuals in the $(t+1)\mathrm{th}$ population, η is the learning rate which means dependence degree on the previous generation. The closer η is to 1, the less a change of distribution is. Especially in case of $\eta = 0$, the distribution is updated based on the population at only the $(t+1)\mathrm{th}$ generation without referring to the past distribution.

$\hat{P}(X_i = x|C_i = c)$ is estimated as follows. At first, S_S individuals are sampled by tournament selection with tournament size T_{edp}, and maximum likelihood estimation is performed based on these selected individuals. Therefore,

$$\hat{P}(X_i = x|C_i = c) = \frac{\#(X_i = x, C_i = c)}{\#(C_i = c)} \quad (5)$$

where $\#(X_i = x, C_i = c)$ is the number of selected individuals that node i is x when its parent node is c, and $\#(C_i = c)$ is the number of selected individuals that the parent node of node i is c.

In most cases, a program tree of a selected individual is smaller than the Bayesian network. Therefore, probabilistic variables in deeper position have fewer samples.

2.4 Program Generation

At first, the acquired distribution $P_t(X_i = x|C_i = c)$ is modified applying Laplace correction [4] by

$$P'_t(X_i = x|C_i = c) = (1-\alpha)P_t(X_i = x|C_i = c) + \alpha P_{bias}(X_i = x|C_i = c) \quad (6)$$

where α is a constant that expresses the Laplace correction rate, $P_{bias}(X_i = x|C_i = c)$ is the probability to bias distribution. For instance, if it is already known that $X_2 =$ " $+$ " is desirable, adjusting $P_{bias}(X_i = x|C_i = c)$ as the probability of $X_2 =$ "$+$" is high would lead to more effective evolution. In this way, the system can incorporate preknowledge by Laplace correction. For our experiments, the Laplace correction rate α is decided as

$$\alpha = 0.01(|F| + |T|) \tag{7}$$

This modification also makes all occurrence probabilities of node symbols positive. Next, according to $P'_t(X_i = x|C_i = c)$, node symbols are decided in sequence from root to terminals. If the size of generated tree reaches d_{max}, only terminal node symbols are selected. Therefore, a larger program tree than the Bayesian network is not generated.

3 Performance of EDP

3.1 Comparative Experiments with GP

The performance was compared for EDP and GP in standard benchmark problems, i.e. a max problem [17], a boolean 6-multiplexer problem [9], and a function regression problem [9]. Let $prog_i$ be a program tree of the $i\mathrm{th}$ individual in a population. If the program tree has some variables, $prog_i(X)$ represents the value obtained by substituting X. If the program tree has no variable, $prog_i$ represents the value returned by the program tree. Let fit_i be the fitness value of the $i\mathrm{th}$ individual.

Max Problem

In a max problem, the purpose is to create the maximum value, based on the assumption that $T = \{0.5\}$ and $F = \{+, *\}$, and the maximum tree depth is 7. For a tree produces the largest value, the $+$ nodes must be used with 0.5 to assemble subtrees A with the value 2.0. These can then be connected via $*$, as shown in Fig. 4. Hence, 65536 is the optimum solution [3]. The fitness value for $i\mathrm{th}$ individual is the value of tree, that is,

$$fit_i = prog_i \tag{8}$$

The parameters of EDP and GP are indicated in Table 1.

Figure 5 and Table 2 show the results of a comparative test using EDP, GP and a random search. The vertical axis represents the max fitness value in a population at each generation: fit_{max}, i.e.

$$fit_{max} = \max_{i \in M} fit_i \tag{9}$$

[3] The maximum node size for the depth of 7 is 127 in a complete binary tree. Within the node size of 127, it is proved that the maximum value is not 65536, but 123596.1914.

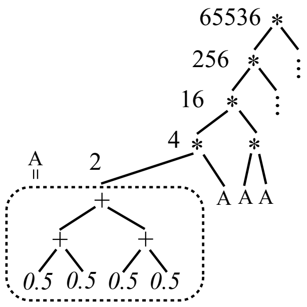

Fig. 4. The maximum value by a tree of limited depth

Table 1. Parameters for a max problem

Common parameters for EDP and GP	
M: population size	200
E_S: elite size	5
F: function node sets	$\{+, *\}$
T: terminal node sets	$\{0.5\}$
P_F: generation probability of function node	$\frac{2}{3}$
P_T: generation probability of terminal node	$\frac{1}{3}$
Tree size limitation in initializing population	max depth = 7
EDP parameters	
α: Laplace correction rate	0.03
P_{bias}: the probability to bias distribution	$\frac{1}{\lvert F\rvert+\lvert T\rvert} = \frac{1}{3}$
η: learning rate	0.2
S_S: sampling size	200
T_{edp}: tournament size for sampling	20
Tree size limitation	max depth = 7
GP parameters	
P_M: mutation probability	0.1
P_C: crossover probability	0.9
T_{gp}: tournament size for GP operator	5
Tree size limitation	max depth = 7

The mean and the standard deviation for 100 runs are indicated in Fig. 5. Note that they are not a mean fitness value and a standard deviation of a population. The solution in an evolutionary computing is given by an individual who has the maximum fitness value in a population. Therefore, system performances should be compared in maximum fitness values. It can be seen that EDP method produces a higher mean fitness value at each generation and also higher performance on the average. In ad-

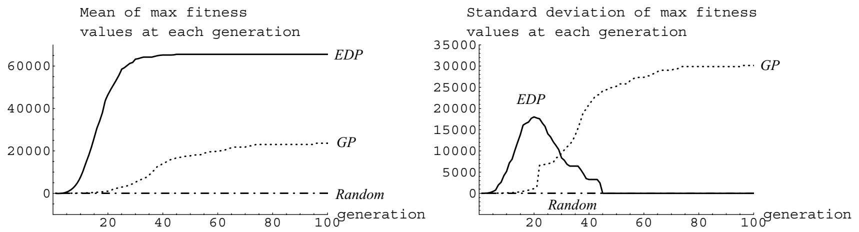

Fig. 5. Comparative results with a max problem

dition, the standard deviation of EDP, i.e. the deviation due to the search runs, is so small that the likelihood of the search being successful is higher.

As presented in Table 2, EDP was able to find the optimal solution in all runs, whereas only 34 runs (out of 100 runs) resulted in evolving the optimal solution with GP. These results suggest intrinsic difference between EDP and GP.

Next, the experiment was carried out with the addition of "0" to the terminal node set. In this problem "0" is completely useless and harmful as a node, and produces non-functional code segments, i.e. introns. As shown in Fig. 6, although the performance of GP was low, with EDP algorithm the most suitable solution was found successfully.

Boolean 6-Multiplexer Problem

Consider the problem of learning the Boolean 6-multiplexer function $F_{6mp} : \{0,1\}^6 \rightarrow \{0,1\}$. The input to the Boolean 6-multiplexer function consists of 2 address bits and 2^2 data bits, where $6 = 2 + 2^2$. The value of the Boolean multi-

Table 2. Percentage of runs finding the optimal solution

Method	Max problem	Multiplexer problem	Max problem adding "0" terminal node
EDP	100	23	86
GP	34	82	0
Random	0	0	0

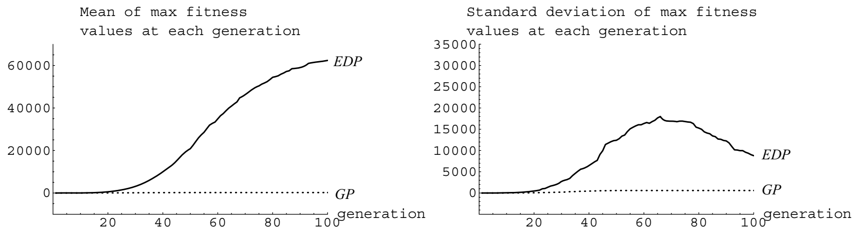

Fig. 6. Comparative results when "0" terminal node was added with a max problem

plexer function is the Boolean value (0 or 1) of the particular data bit that is singled out by the 2 address bits of the multiplexer. Formally,

$$F_{6mp}(a_0, a_1, d_0, d_1, d_2, d_3) = d_{2^1 a_1 + a_0} \tag{10}$$

The node set is $T = \{x_0, x_1, x_2, x_3, x_4, x_5\}$, $F = \{and, or, not\}$. The parameters of EDP and GP are indicated in Table 3. There are $2^6 = 64$ possible combinations of the 6 arguments, and we use the entire set of 64 combinations of arguments as the fitness cases for evaluating fitness. That is, we do not use sampling. The fitness values are simply the number of fitness cases for which the individual tree returns a correct Boolean value. Let X_i be an input data set, i.e. $X_j = \{x_{j1}, \ldots, x_{j6}\}$, where x_{jk} is the $k\mathrm{th}$ digit of the number j. Then, the fitness value is given with the following formula:

$$fit_i = \sum_{j=0}^{63} \mathrm{match}(prog_i(X_j), F_{6mp}(X_j)) \tag{11}$$

where

Table 3. Parameter for a boolean 6-multiplexer problem

Common parameters for EDP and GP	
M: population size	500
E_S: elite size	5
F: function node sets	$\{and, or, not\}$
T: terminal node sets	$\{x_0, x_1, x_2, x_3, x_4, x_5\}$
P_F: generation probability of function node	$\frac{3}{9}$
P_T: generation probability of terminal node	$\frac{6}{9}$
Tree size limitation in initializing population	max depth = 6
EDP parameters	
α: Laplace correction rate	0.09
P_{bias}: the probability to bias distribution	$\frac{1}{\lvert F\rvert+\lvert T\rvert} = \frac{1}{9}$
η: learning rate	0.2
S_S: sampling size	200
T_{edp}: tournament size for sampling	20
Tree size limitation	max depth = 6
GP parameters	
P_M: mutation probability	0.1
P_C: crossover probability	0.9
T_{gp}: tournament size for GP operator	5
Tree size limitation	max depth = 6

$$\mathrm{match}(a,b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{else} \end{cases} \tag{12}$$

Figure 7 shows the results of a comparative test using EDP, GP and a random search. We cannot confirm the superiority of EDP with this experiment. In 6-multiplexer problem, EDP could not search more efficiently than GP. However, EDP was superior to a random search.

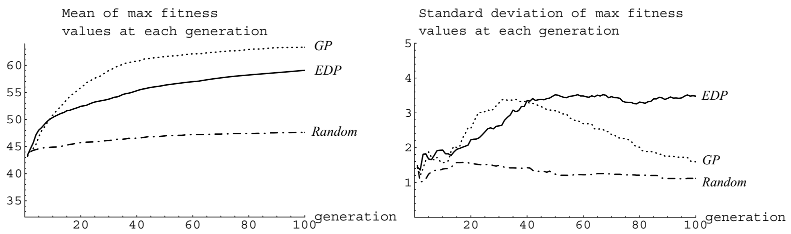

Fig. 7. Comparative results with a boolean 6-multiplexer problem

Function Regression Problem

Consider a function regression problem. f_{obj} is the function to be approximated. The fitness value is given with the following formula:

$$fitness = 1000 - 50 \sum_{j=1}^{30} |prog(X_j) - f_{obj}(X_j)| \tag{13}$$

where

$$X_j = 0.2(j-1) \tag{14}$$

i.e. training examples are the real values at intervals of 0.2 from 0 to 5.8. Objective functions are

$$\text{A} : f_{obj}(x) = (2 - 0.3x)\sin(2x)\cos(3x) + 0.01x^2 \tag{15}$$

$$\text{B} : f_{obj}(x) = x\cos(x)\sin(x)(\sin^2(x)\cos(x) - 1) \tag{16}$$

$$\text{C} : f_{obj}(x) = x^3\cos(x)\sin(x)e^{-x}(\sin^2(x)\cos(x) - 1) \tag{17}$$

which are plotted in Fig. 8. Objective function C is cited from [19]. Although B is obtained from simplification of C, B is more difficult to search. A is our original function and the most difficult of the three objective functions.

As indicated in Figs. 9, 10 and 11, EDP's performance was worse than GP's in a function regression problem. This result seems to suggest that EDP is not always superior. However, as we can see later, the EDP operator plays an inevitable role in combination with GP. The effectiveness of the hybrid system of EDP and GP is described in Sect. 4.

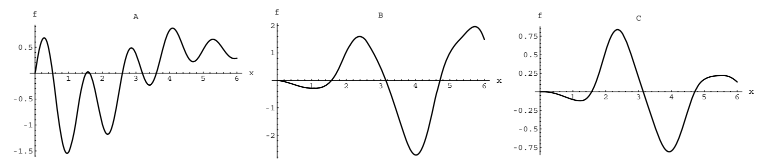

Fig. 8. Objective functions

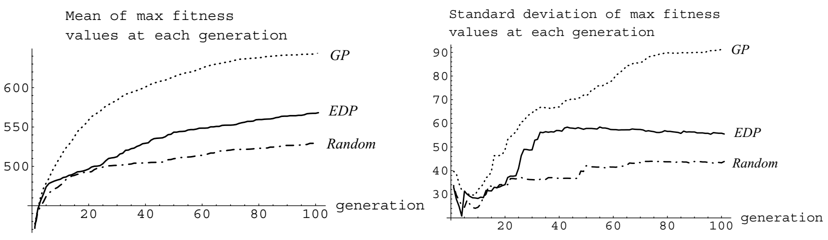

Fig. 9. Comparative results with objective function A

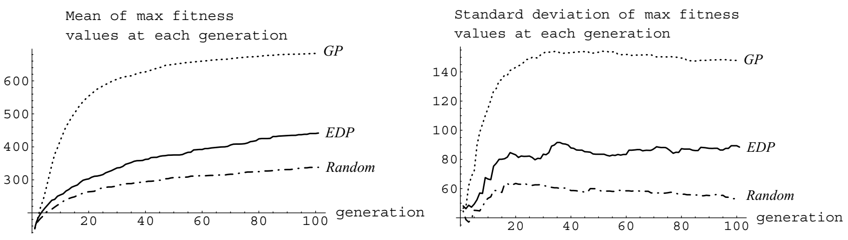

Fig. 10. Comparative results with objective function B

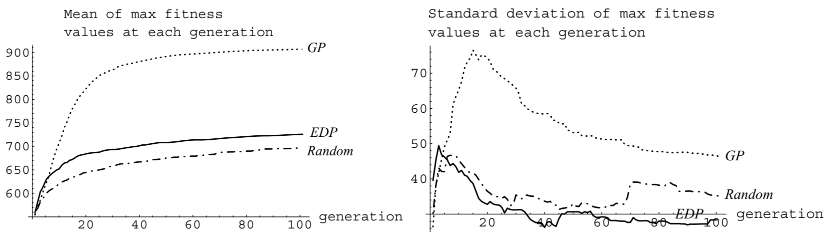

Fig. 11. Comparative results with objective function C

3.2 Summaries of EDP Performance

EDP was able to search for a solution effectively in a GP-hard problem, i.e. a max problem. On the other hand, in both a boolean 6-multiplexer problem and a function regression problem, it has been shown that EDP's performance was worse than GP's. In order to conclude that the differences of these values are statistically significant and reliable, not only mean but also standard deviation and sample size (100) should be taken into consideration. We used Welch's test for the obtained experimental re-

Table 4. Parameter for a function regression problem

Common parameters for EDP and GP	
M: population size	1000
E_S: elite size	5
F: function node sets	$\{+, -, *, /, \cos, \sin\}$
T: terminal node sets	$\{x, 0.05, 0.10, 0.15, \ldots, 1.00\}$
P_F: generation probability of function node	0.8
P_T: generation probability of terminal node	0.2
Tree size limitation in initializing population	max depth = 6

EDP parameters	
α: Laplace correction rate	0.27
P_{bias}: the probability to bias distribution	$\frac{1}{\lvert F\rvert+\lvert T\rvert} = \frac{1}{27}$
η: learning rate	0.2
S_S: sampling size	200
T_{edp}: tournament size for sampling	20
Tree size limitation	max depth = 6

GP parameters	
P_M: mutation probability	0.1
P_C: crossover probability	0.9
T_{gp}: tournament size for GP operator	5
Tree size limitation	max depth = 6

Table 5. P-values on Welch's test

Problem	EDP and GP	EDP and Random
Max problem	3.49×10^{-25}	1.10×10^{-340}
Multiplexer problem	4.53×10^{-21}	2.52×10^{-59}
Regression A	2.53×10^{-11}	1.80×10^{-7}
Regression B	8.52×10^{-30}	5.44×10^{-19}
Regression C	6.96×10^{-75}	7.03×10^{-10}

sults. By means of Welch's test, it can be judged whether 2 data sets are samples from the same statistical population or not. As a result of Welch's test with 5% significance level, the differences between EDP and GP at the $100\mathrm{th}$ generation were significant in all cases. Statistically speaking, the null hypothesis that data in EDP and in GP were sampled from the same statistical population was rejected (the probability that the null hypothesis is correct is less than 5%). Welch's test concluded that the differences were significant. Table 5 indicates the p-values obtained in the test. This seems to indicate that EDP works intrinsically differently from the traditional GP.

In a max problem, in order to produce better solutions, it is necessary for EDP to increase the generation probability of "$*$" from the depth 1 to 4, and the probability of "$+$" at the depth 6. In the early stage of the evolution, the generation probability of "$+$" is expected to become high in a shallow part. Then, more frequently subtrees identical to $(+\ (+\ 0.5\ 0.5)\ (+\ 0.5\ 0.5))$ (subtree A shown in Fig. 4) are produced in a deep part, the higher the generation probability of "$*$" becomes.

In a boolean 6-multiplexer problem, a positional restriction of EDP operator seems to have caused the worse performance. Using the 3-multiplexer function F_{3mp}, it is easy to compose the 6-multiplexer function in the following way:

$$\begin{aligned} F_{6mp}(a_0, a_1, d_0, d_1, d_2, d_3) = \\ &(or\ (and\ F_{3mp}(a_1, d_0, d_1)\ (not\ a_0)) \\ &\quad (and\ F_{3mp}(a_1, d_2, d_3)\ a_0)) \end{aligned} \tag{18}$$

Furthermore, it has been reported that the 11-multiplexer function and the 6-multiplexer function were easily acquired by GP with the 6-multiplexer and the 3-multiplexer structures respectively [9]. An individual equivalent to the 3-multiplexer function would be assigned a high fitness value, i.e. $32 + 16 = 48$. Therefore, the composition of the 3-multiplexer functions is so important for the effective evolution of the 6-multiplexer function that they are expected to prosper in a population. Note that useful subtrees, i.e. so-called building blocks, cannot shift their position with EDP because the probability distribution is dependent on the position within a tree, while GP crossover can move them to an arbitrary position. In other words, EDP imposes a positional restriction. Consequently, EDP could not always use the generated structure of the 3-multiplexer function efficiently in order to compose the 6-multiplexer function. This is the reason why EDP operator failed to generate better individuals in some cases.

4 Hybrid System of EDP and GP

4.1 Algorithm of Hybrid System

We research the hybrid system which consists of EDP and GP. Figure 12 indicates a pseudo code of our hybrid system.

The most important parameter in this algorithm is "r", it decides the system behavior and the ratio of GP to EDP in an individual generation, called the hybrid ratio. Through the combination of EDP and GP, the difficulty indicated in Sect. 3.2 might be overcome. However, it is not obvious whether GP gains anything from hybridization. In this section, we test the system performance in a function regression problem changing the hybrid ratio r from 0 to 1.

4.2 Performance Difference Due to the Hybrid Ratio

Figures 13, 14, and 15 show the mean of max fitness values for 100 runs. Note that it is not a mean fitness value of a population, but a mean value of the maximum fitness value.

Let P be a population, S a set of selected individuals, D a distribution, r a hybrid ratio, E_S an elite size, and M a population size.

```
1. $P$ := Generate_Programs_Randomly
2. While (True)
3.     Evaluate_Individuals($P$)
4.     If (termination_criterion) Return($P$)
5.     $S$ := Selection($P$)
6.     $D$ := Estimate_Distribution($S$)
7.     $P$ := Elite_Selection($P$, $E_S$)
8.     $P$ := P + Crossover&Mutation($P$, $rM - E_S$)
9.     $P$ := P + Generate_Individuals($D$, $(1 - r)M$)
```

Fig. 12. Pseudo code of the hybrid system

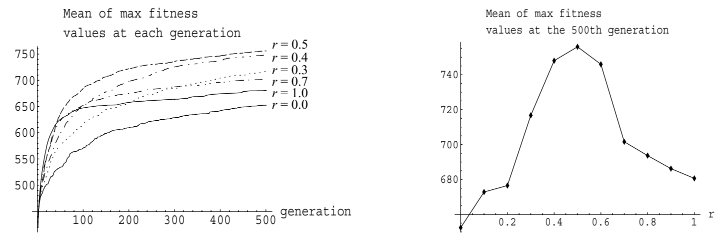

Fig. 13. Results for objective function A

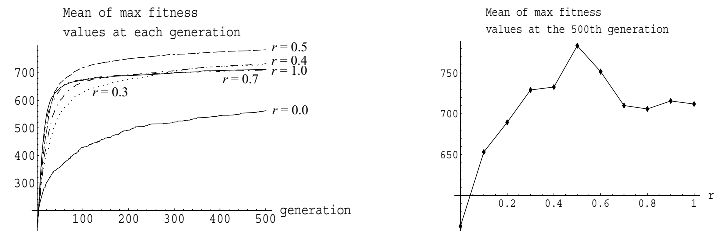

Fig. 14. Results for objective function B

Figure 16 shows the frequency of runs in which the maximum fitness value at the $500\mathrm{th}$ generation is over x, that is,

$$F(x) = \sum_{k=1}^{100} \delta(x \leq f_{max}{}_{k,500}) \tag{19}$$

where $f_{max}{}_{k,500}$ is the maximum fitness value in a population of the $500\mathrm{th}$ generation at the $k\mathrm{th}$ run, and

$$\delta(x \leq a) = \begin{cases} 1 & : x \leq a \\ 0 & : x > a \end{cases} \tag{20}$$

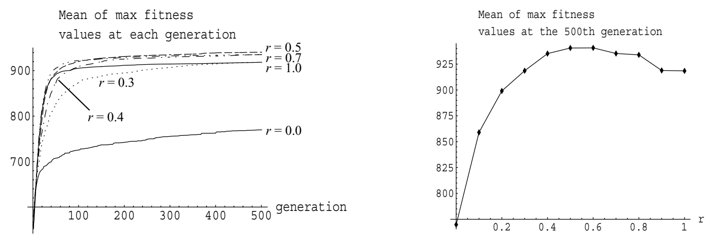

Fig. 15. Results for objective function C

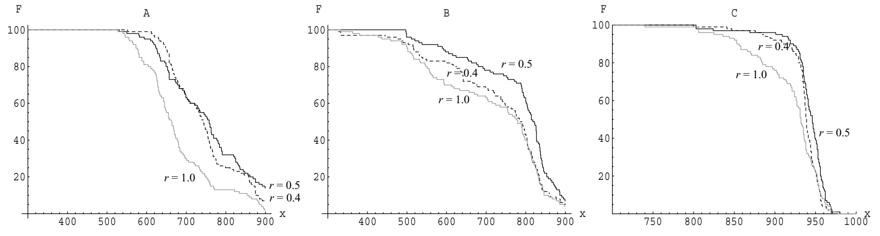

Fig. 16. $F(x)$: frequency of max fitness at the $500\mathrm{th}$ generation greater than x, with objective functions A and B

Figures 13, 14, 15, and 16 indicate the similar tendency in each case. Although the $r = 1.0$ system which is pure GP, demonstrated the best performance in younger generations, gradually hybrid systems overtook pure GP one after another. The "overtaking" was conspicuous when $r = 0.3$ or $r = 0.4$. At the $500\mathrm{th}$ generation, the performance of the $r = 0.5$ system was the best in all cases. The system performances at the $500\mathrm{th}$ generation reached a peak at $r = 0.5$, and got worse as the hybrid ratio was biased.

As a result of Welch's test with 5% significance level, the differences between the $r = 0.5$ system and pure GP at the $500\mathrm{th}$ generation were significant in all cases. The p-values obtained in the test for objective function A, B, and C were 2.57×10^{-7}, 1.23×10^{-4}, and 1.52×10^{-27} respectively. In the case of objective function C, although the difference in values was slight, standard deviation was negligible (see Fig. 16); Welch's test concluded that the differences were significant.

Mean cannot give adequate information for system performances, hence we showed Fig. 16. Figure 16 demonstrates that the hybrid system is also superior to pure GP in the success rate of a search. For instance, in the case of A, the probabilities that the maximum fitness value at the $500\mathrm{th}$ generation is over 700 are $\frac{63}{100}$ with $r = 0.5$ and $\frac{30}{100}$ with pure GP respectively.

4.3 Analysis of the Behavior of EDP

This section investigates the hybrid system's performance, changing the hybrid ratio r at each generation. In Fig. 13, until the $50\mathrm{th}$ generation, the higher the GP ratio of the system is, the better its performance. Therefore, the system that has a high GP

Table 6. Systems with changing r, where i is the generation number

System	r
A: classical hybrid	$r = 0.3$
B: classical hybrid	$r = 0.5$
C: pure GP	$r = 1.0$
D: linear increasing	$r = \frac{i}{500}$
E: linear decreasing	$r = 1 - \frac{i}{500}$
F: random	r is a random value at each generation
G: switching	$r = \begin{cases} 1.0 & : i < 205 \\ 0.3 & : i \geq 205 \end{cases}$
H: switching	$r = \begin{cases} 1.0 & : i < 40 \\ 0.5 & : i \geq 40 \end{cases}$

ratio in younger generations and decreases the ratio later is expected to have higher performance.

Comparative experiments were carried out with 8 variations of systems, as shown in Table 6. The objective function is the first one used in Sect. 3.1, i.e. (15). In the system D, the GP ratio is linearly increased from 0, at the initial generation, to 1.0, at the $500\mathrm{th}$ generation, whereas it is linearly decreased in the system E. In the system G, the ratio is changed from 1.0 to 0.3 at the $205\mathrm{th}$ generation. Note that the $r = 0.3$ system overtook the pure GP at the $205\mathrm{th}$ generation (see Fig. 13). In the system H, the ratio is tuned in the same manner as G. Therefore, H and G are supposed to be the top favorites among these systems.

Figures 17 and 18 show the results of comparative experiments. Surprisingly, system A overtook G. As a result of Welch's test with 5% significance level, the differences were significant. The p-value obtained in the test was 0.026. This result means that population states of A and G are far different in spite of close performance at the $205\mathrm{th}$ generation. In other words, EDP's behavior before the $205\mathrm{th}$ generation likely has a good influence later. Although B also overtook H, the result was not significant statistically. The p-value obtained in the test for system B and H was 0.364.

Another interesting result is that system D was superior to all other systems, especially E. As a result of Welch's test with 5% significance level, the differences were significant. The p-value was 0.0473. Although it was expected that D would be worse than E, judging from Fig. 13, the result was quite the opposite. This point is evidence that EDP functions well in early generations.

In order to test the hypothesis that the probability distribution memorizes the past EDP's work, the system of $\eta = 0$ was simulated. This system estimates distribution without referring to the past distribution (see Sect. 2.3). Objective function A was used.

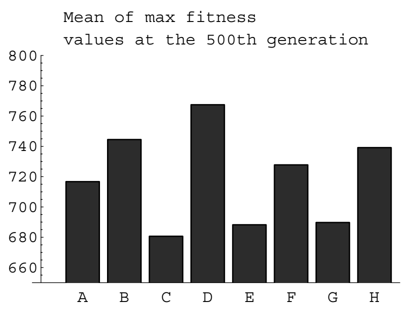

Fig. 17. Mean of max fitness values at the $500\mathrm{th}$ generation

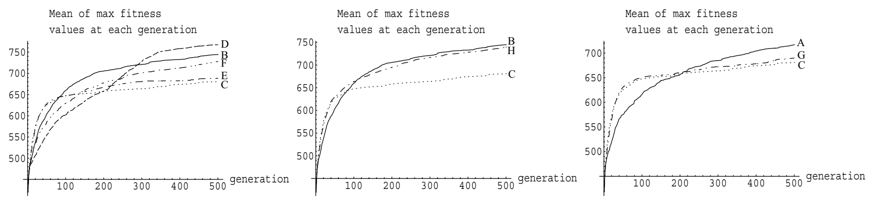

Fig. 18. Mean of max fitness values at each generation

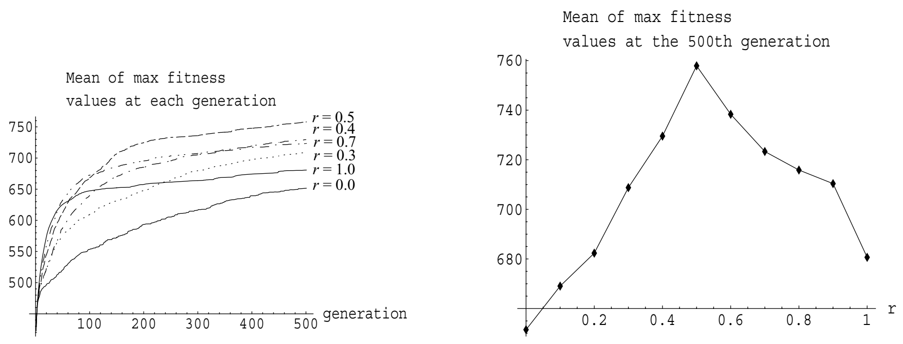

Fig. 19. System of $\eta = 0$

As indicated in Fig. 19, the characteristic of the hybrid system was kept. The "overtaking" still took place and the $r = 0.5$ system was the best. Therefore, the past information accumulated in the probability distribution does not cause the high performance of the hybrid system.

5 Discussion

The previous experimental results revealed the following aspects of EDP:

- EDP's search was intrinsically different from GP's.

- EDP's search was successful in a max problem with the addition of “0” to the terminal node set.
- Subtrees were not easily shifted in EDP.
- The hybrid system outperformed the pure GP in later generations.
- The hybrid system with linearly increasing hybrid ratio gave the best performance.

EDP does not refer to the previous generation directly, but abandon all individuals in previous generation and generate new individuals based on the distribution at an every generation. Thus, a random search is regarded as EDP with an uniform distribution. In 6-multiplexer problem and a regression problem, although EDP could not search more efficiently than GP, EDP was superior to a random search. Therefore, the probability distribution could be estimated effectively. The estimation of a distribution was done to some extent for the program search.

In the $r = 0.5$ hybrid system, the updating times of the maximum fitness values at each generation of the EDP operator and the GP operator are counted respectively. Surprisingly, the EDP operator hardly contributes to construction of the best individual directly, and only the GP operator does. In addition, as shown in Fig. 17, system D, which has linearly increasing hybrid ratio, gave the best performance of all. System D cannot benefit from EDP in later generations. These results suggest individuals constructed by EDP have more multifarious sub-structures in an early stage, and these various structures are put together in later generations. It is GP that can build better individuals, but not EDP.

The hybrid algorithm was tested in a function regression problem where the behavior of the EDP algorithm was bad. We also research how the hybrid system degrades in a max problem where previously EDP behaved properly. Figure 20 shows the performance of the hybrid system in a max problem. Although the performance of the hybrid system was a little worse than pure EDP's, the search by the hybrid system was successful.

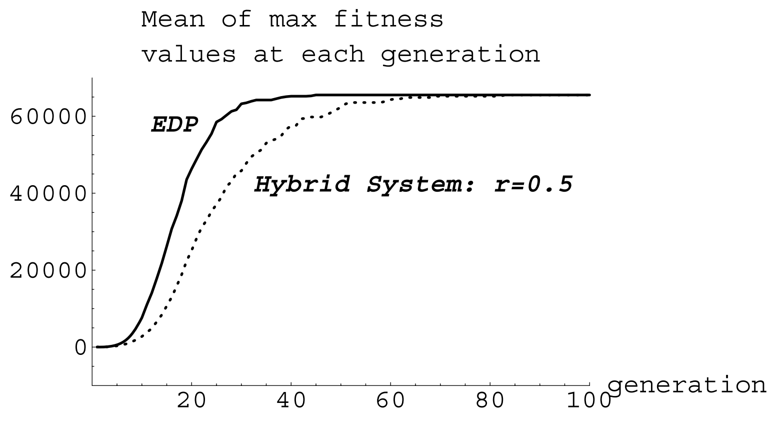

Fig. 20. Performance of the hybrid system in a max problem

6 Conclusion

This paper presented a new EDA-based approach, i.e. EDP, to program evolution and have shown the experimental results with EDP and GP.

When "0" was added to a set of terminal nodes, EDP performed much better than GP. We cannot always know what are effective nodes for problems before. This result suggests that EDP can perform evolution skillfully even if harmful nodes are included in a node set. Thus, it is expected that the occurrence probability of this harmful node is kept lower by the EDP method due to the obtained distribution. This indicates that EDP can control introns effectively, while GP may suffer from increasing introns and allow them to cause a bloat [10].

The experimental results clearly indicated that EDP worked effectively in early generations and contributed to later high performance. It turned out that pure GP could not generate enough kinds of subtrees in early generations to build better solutions. On the other hand, useful subtrees are not easily shifted by EDP to another position in the tree. We conclude that hybridization helps EDP and GP compensate for their defects and build a better evolutionary system.

Future and Related Works

Probabilistic Incremental Program Evolution (PIPE) [19] was used to perform a program search based on a probabilistic model. However, PIPE assumes the independence of program nodes and differs from our approach using a Bayesian network in this assumption. The merits of having probabilistic dependency relationship are as follows:

1. Because an occurrence probability of a node symbol is dependent on its parent node, estimation and generation are serial from a parent node to a child. Therefore, it can derive and generate building blocks.
2. The past dominant structure can survive after switching the probability distribution based on a parent node symbol.

On the other hand, optimization using a Bayesian network is much researched, e.g., EBNA (Estimation of Bayesian Network Algorithm) [12] and EGNA (Estimation of Gaussian Networks Algorithm) [13]. Recently, EDA has been extended with reinforcement learning [14]. We are also currently working on EDA application for a gene expression-based classification [15]. However, their application is limited to fixed length array search problems, not program search.

It is not clear how EDP really works in the hybrid system. In future works, the details of EDP's facilities in early generations will be researched. We are also interested in the control rule of the hybrid ratio r and the robust behavior shown in our experiments.

The Bayesian network in our probabilistic model has the simplest topology, i.e. only parent-child links exist. The model selection is one of the most important problems. As the number of dependent variables per a variable increases, the required memory size is exponentially increasing. The adequate sampling size for updating a distribution is also proportional to the exponential of the number of dependency links per a node. Therefore, the trade-off exists between the performance and calculation costs. Our future research will be on the study of the system performance with other topologies. We also plan to improve EDP in order to shift subtrees within a program tree, independently from the hybridization with GP.

This chapter discussed the program evolution on the premise that program representation consists of a single parse tree. However, the validity of the representation depends on the problem class. Without recursion and memory, the expressiveness of a parse tree is not Turing-complete. It is suggested that the different choice of representation will result in the different program evolution [20]. The extension of the program representation should be considered for the sake of establishing a probabilistic model-based evolution.

References

1. P. J. Angeline. Two selfadaptive crossover operations for genetic programming. In *Advances in Genetic Programming II*. MIT Press, 1995.
2. P. J. Angeline. Subtree crossover causes bloat. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pp. 745–752. Morgan Kaufmann, 22–25 July 1998.
3. S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, PA, 1994.
4. B. Cestnik. Estimating probabilities: A crucial task in machine learning. In *Proceedings of the 9th European Conference on Artificial Intelligence*, pp. 147–149, 1990.
5. J. M. Daida, H. Li, R. Tang, and A. M. Hilss. What makes a problem GP-hard? validating a hypothesis of structural causes. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-2004*, pp. 1665–1677. Springer-Verlag, 2003.
6. H. Iba and H. de Garis. Extending genetic programming with recombinative guidance. In *Advances in Genetic Programming 2*, pp. 69–88. MIT Press, 1995.
7. C. Igel and K. Chellapilla. Investigating the influence of depth and degree of genotypic change on fitness in genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-1999*, volume 2, pp. 1061–1068. Morgan Kaufmann, 13-17 July 1999.
8. T. Ito, H. Iba, and S. Sato. Depth-dependent crossover for genetic programming. In *Proceedings of the 1998 IEEE World Congress on Computational Intelligence*, pp. 775–780. IEEE Press, 5-9 May 1998.
9. J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.
10. W. B. Langdon. Size fair and homologous tree genetic programming crossovers. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-1999*, volume 2, pp. 1092–1097. Morgan Kaufmann, 13-17 1999.
11. W. B. Langdon and R. Poli. Genetic programming bloat with dynamic fitness. In *Proceedings of the First European Workshop on Genetic Programming*. Springer-Verlag, 1998.
12. P. Larrañaga, R. Etxeberria, J. A. Lozano, and J. M. Peña. Combinatorial optimization by learning and simulation of Bayesian networks. In *Proceedings of the Conference in Uncertainty in Artificial Intelligence: UAI-2000*, pp. 343–352, 2000.

13. P. Larrañaga, R. Etxeberria, J. A. Lozano, and J. M. Peña. Optimization in continuous domains by learning and simulation of Gaussian networks. In *Proceedings of the Workshop in Optimization by Building and Using Probabilistic Models*, pp. 201–204, 2000.
14. T. K. Paul and H. Iba. Reinforcement learning estimation of distribution algorithm. In *Proceedings of Genetic and Evolutionary Computation Conference GECCO-2003*. Springer-Verlag, 2003.
15. T. K. Paul and H. Iba. Selection of the most useful subset of genes for gene expression-based classification. In *Proceedings of Congress on Evolutionary Computation: CEC-2004*, 2004.
16. R. Poli and W. B. Langdon. On the search properties of different crossover operators in genetic programming. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pp. 293–301. Morgan Kaufmann, 22–25 1998.
17. R. Poli and W. B. Langdon. *Foundations of Genetic Programming*. Springer-Verlag, 2002.
18. W. F. Punch, D. Zongker, and E. D. Goodman. The royal tree problem, a benchmark for single and multiple population genetic programming. In *Advances in Genetic Programming II*, pp. 299–316. MIT Press, 1995.
19. R. P. Salustowicz and J. Schmidhuber. Probabilistic incremental program evolution: Stochastic search through program space. In M. van Someren and G. Widmer, editors, *Machine Learning: ECML-97*, volume 1224, pp. 213–220. Springer-Verlag, 1997.
20. T. Yabuki and H. Iba. Genetic programming using a Turing complete representation: recurrent network consisting of trees. In L. Nunes de Castro and F. J. Von Zuben, editors, *Recent Developments in Biologically Inspired Computing*. Idea Group Inc., 2004. (to be published).
21. K. Yanai and H. Iba. Estimation of distribution programming based on Bayesian networks. In *Proceedings of Congress on Evolutionary Computation: CEC-2003*, pp. 1618–1625, 2003.