Estimation of Bayesian Network for Program Generation

Yoshihiko Hasegawa and Hitoshi Iba

Dept. Frontier Informatics, Graduate School of Frontier Sciences,
The University of Tokyo, Japan
{hasegawa,iba}@iba.k.u-tokyo.ac.jp

Abstract. Genetic Programming (GP) is a powerful optimization algorithm, which employs crossover for a main genetic operator. Because a crossover operator in GP selects sub-trees randomly, the building blocks may be destroyed by crossover. Recently, algorithms called PMBGPs (Probabilistic Model Building GP) based on probabilistic techniques have been proposed in order to improve the problem above. We propose a new PMBGP employing Bayesian network for generating new individuals with a special chromosome called *expanded parse tree*, which much reduces the number of possible symbols at each node. Although the large number of symbols gives rise to the large conditional probability table and requires a lot of samples to estimate the interactions among nodes, a use of the expanded parse tree overcomes these problems. A computational experiment on a deceptive MAX problem (DMAX problem) demonstrates that our new PMBGP is superior to other program evolution methods.

1 Introduction

In this paper, we propose a program optimization algorithm based on GP which adopts a probabilistic model for a genetic operator. Our approach POLE (Program Optimization with Linkage Estimation) [1] employs Bayesian network for a probabilistic model and uses a special chromosome called *expanded parse tree* to overcome the problem of prior PMBGPs.

GP is an extended algorithm of GA and is capable of handling programs and functions. Because GP has a structural chromosome which is very flexible, GP has been applied to many problems (robot engineering, financial engineering, bio informatics, etc) and has been considered to be a very powerful way for solving these problems.

The main genetic operator in Evolutionary Algorithms (EAs) is a crossover operator. Although the crossover operator is a very powerful operator in GA and GP, it also has a drawback. In simple GA and GP, crossover is carried out randomly regardless of the problem it applied. As a result, the building blocks may be destroyed by crossover and this destruction makes the search inefficient.

Recently new genetic operators based on probabilistic techniques have been proposed to replace the old crossover operator. These probabilistic techniques are considered to be able to overcome the drawbacks of the crossover operator. More and more attentions have been paid to these algorithms. It is called PM-BGA (Probabilistic Model Building GA) [2] or EDA (Estimation of Distribution Algorithm) [3], and PMBGP (Probabilistic Model Building GP) [4].

In the prior PMBGPs based on the standard GP, several probabilistic models have been applied (e.g. univariate, parent-child relation). In order to express the general probabilistic interactions, an estimation of Bayesian network is a good solution. However, there is a big problem on applying the estimation of Bayesian network to the program evolution. Because GP uses many symbols during the search (e.g. $F = \{+, -, \times, ...\}$, $T = \{x, y, \Re, ...\}$), the number of possible symbols at each node become very large. When the number of possible symbols is large, conditional probability tables (CPT) which express the quantitative relationship among nodes becomes large. As a result, many samples are required to construct the Bayesian network. Furthermore, all the programs generated with Bayesian network have to be syntactically correct.

In order to overcome the problem explained above, our approach adopts a special chromosome called expanded parse tree (see Section 3.1) for expressing the individuals. With the expanded parse tree, the number of possible symbols can be dramatically reduced. At the same time, all program generated under the expanded parse tree are syntactically correct.

Characteristics of our approach are listed below.

- The Bayesian network is used for estimating the interactions among nodes.
- A special chromosome called expanded parse tree is employed.

In the following sections, we describe the details of our approach.

2 Related Works

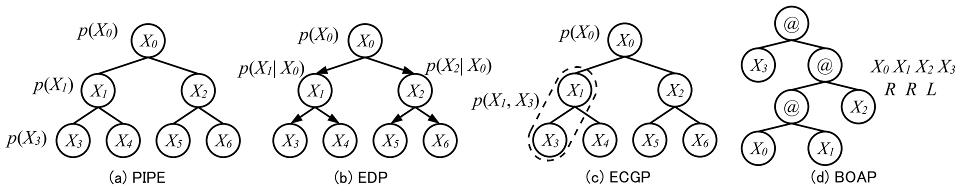

Fig. 1. Graphical interpretation of PMBGPs.

EAs based on probabilistic models were first carried out in the field of binary GA. Binary PMBGAs can be classified into three groups: univariate model, bi-variate model and multivariate model. In these fields, many algorithms have been proposed. Especially, multivariate algorithms which use Bayesian network (e.g.

BOA (Bayesian Optimization Algorithm) [5], EBNA (Estimation of Bayesian Network Algorithm) [6], LFDA (Learning Factorized Distribution Algorithm) [7], etc) are considered to be very strong algorithms because of their model flexibility.

PMBGPs can be broadly classified into two groups: the algorithms based on simple GP and those based on grammar-guided GP.

- Standard GP based PMBGP (Figure 1)
  PIPE (Probabilistic Incremental Program Evolution) [8], EDP (Estimation of Distribution Programming) [9], ECGP (Extended Compact Genetic Programming) [4], BOAP (BOA Programming) [10]
- Grammar Model based PMBGP
  SG-GP (Stochastic Grammar-based GP) [11], PEEL (Program Evolution with Explicit Learning) [12], GMPE (Grammar Model based Program Evolution) [13], Grammar Transformations in EDA [14], BAP (Bayesian Automatic Programming) [15]

Our approach POLE, which belongs to the standard GP group, is much improved over the prior models (PIPE, EDP, ECGP, BOAP). Because PIPE and EDP use fixed probabilistic models, these models are less flexible compared to Bayesian network. Although ECGP can estimate any interactions, the size of joint distribution becomes very large and only small building blocks can be estimated. In the *zigzag tree* used in BOAP, syntactically incorrect individuals can be easily generated and these incorrect individuals have to be managed with some heuristics. Because our approach employs Bayesian network and the expanded parse tree, any interactions can be estimated with smaller conditional probability table size. Furthermore, all the programs generated with our approach are syntactically correct.

3 Proposed Method

PIPE, EDP and ECGP use the standard parse tree as simple GP. In these algorithms, all programs are handled as a full α-ary tree (Figure 2(a)), α being the maximum arity among functions. The positions of each node are superposed and the probability distributions are constructed by counting the frequencies. In the standard parse tree, both functions and terminals are located at branches in the tree. Although terminal z and function *exp* in Figure 2(a) are at the same depth, the types of these nodes are different. The size of traditional CPT is calculated with Equation 1, which also represents the number of free parameters in Bayesian network.

$$K = \sum_{i=0}^{n-1} \|\Pi_i\| \left(\|X_i\| - 1\right) \tag{1}$$

In the case of the standard parse tree, $\|X_i\| = |F \cup T|$, $\|\Pi_i\| = |F \cup T|^{|\Pi_i|}$, where $\|X_i\|$ is the number of possible instances of X_i, Π_i represents the set of parent nodes of X_i in Bayesian network and n denotes the number of nodes. Usually, GP uses about 10 symbols during the search. Then $\|X_i\| = |F \cup T| = 10$. On the other hand, binary GA uses only 2 symbols, $\|X_i\| = 2$. This example shows that the size of CPT in PMBGP based on the standard parse tree is much larger compared to that in binary PMBGA.

In our approach, BIC (Bayesian Information Criteria) [16] is used for scoring networks. In this metric, CPT size K is used for a penalty term. If the size of CPT is large, only small number of relations can be estimated. Furthermore, more samples are required to construct the Bayesian network.

In order to improve the problems mentioned above, we propose the use of the expanded parse tree with Bayesian network. This expression much reduces the number of possible symbols at each node. As a result, size of CPT can be dramatically reduced. We will show the reason of the size reduction in the next section.

3.1 Expanded Parse Tree

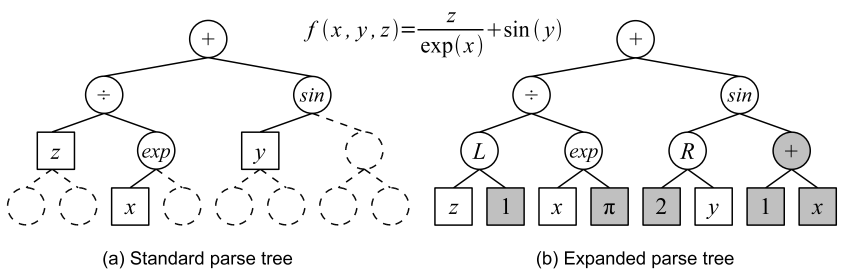

Fig. 2. Standard parse tree (a) and the expanded parse tree (b). The gray nodes represent introns. Squares and circles represent functions and terminals. Both trees are syntactically identical.

Expanded parse tree is an expression way for tree structures (programs, functions, etc) and was proposed in an algorithm named Evolutionary Programming with Introns (EP-I) [17]. The expanded parse tree was originally devised to apply GA crossover to GP. The use of the expanded parse tree enables application of GA operators to GP.

GP and GA are based on the same concept of natural evolution and have a lot in common. GA uses linear arrays and crossover is done by swapping the sub-arrays of two individuals. On the other hand, GP adopts tree structures and crossover is operated by exchanging sub-trees. Tree structures can be converted to linear arrays (and also fixed length by inserting *null* nodes) as GA. However, the node types (function or terminal) of each position differ among individuals and this difference gives rise to the syntactic destruction after applying GA-style crossover to GP.

If the length of chromosomes is fixed and the node types (function of terminal) at each position are fixed among all individuals, GA crossover can be applied. EP-I extended the standard parse tree to satisfy the above request by inserting the selector nodes and adding introns beneath unused arguments. In EP-I, this extended expression is called *expanded parse tree*. Figure 2(b) describes an example of the expanded parse tree. This expanded parse tree is syntactically identical to Figure 2(a). Figure 2(a) expresses the standard parse tree used in the traditional GP. Expanded parse tree is expressed with full α-ary tree (α is the maximum arity among functions). The expanded parse tree inserts selector operators (L and R) to the parse tree in order to make the terminal nodes positioned at the leaves of the full α-ary tree. In this figure, L and R denote selector operator and are operated as $R(x, y) = y$ and $L(x, y) = x$. However with these extra operators, the length of individuals are different among individuals because there are operators which have fewer arity than the maximum arity (In Figure 2(b), sin has only one arity and this is fewer than 2 which is the maximum arity). EP-I attaches introns to the unused parameters for the sake of overcoming the problem. For instance, exp is converted to 2 arity function and evaluated $exp(x, y) = exp(x)$. In Figure 2(b), gray nodes stand for introns. Introns will not be interpreted during evaluation. The standard parse tree of maximum depth D whose maximum arity is α can be converted to a full α-ary expanded parse tree of maximum depth D.

In the expanded parse tree, node types at each position are identical among all individuals (Figure 2(b)). In Figure 2, nodes represented with circles are functions and squares are terminals. We can see all function nodes are positioned at the branches and all terminals at leaves. Then the GA-style crossover can be applied to the expanded parse tree.

In the previous section, we have denoted that the size of CPT is very large when applying the Bayesian network to the standard parse tree. This is because we have to consider the possibility of both F and T in each position in the standard parse tree. However, in the expanded parse tree which is used in our approach, function nodes are always positioned at the branches, and the terminal node at the leaves (Figure 2(b)). With this property, CPT size can be shrunk.

We will show the reduction of CPT size in the expanded parse tree with an example. Let us assume the parity problem ($F = \{And, Or, Nand, Nor\}$, $T = \{d_0, d_1, d_2, d_3, d_4, d_5\}$). Then we consider the interactions between nodes positioned at branches (Figure 3). In the case of the standard parse tree, the number of possible instances at the branches is $|F \cup T| = 10$. CPT size becomes $K = 90$ with Equation 1. On the other hand, in the expanded parse tree case, CPT size becomes $K = 20$ because only function nodes and a selector node L are positioned at the branches. This example shows that CPT size in the expanded parse tree is $2/9$ times as small as that in the standard parse tree. This difference becomes larger when the node has more parent nodes. We have denoted that

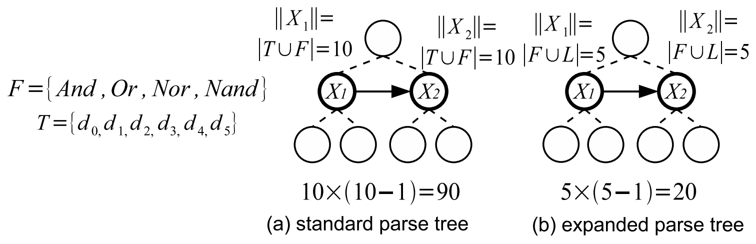
$\|X_1\| = |T \cup F| = 10$
$\|X_2\| = |T \cup F| = 10$
$\|X_1\| = |F \cup L| = 5$
$\|X_2\| = |F \cup L| = 5$
$F = \{And, Or, Nor, Nand\}$
$T = \{d_0, d_1, d_2, d_3, d_4, d_5\}$
X1
X2
$10 \times (10-1) = 90$
$5 \times (5-1) = 20$
(a) standard parse tree
(b) expanded parse tree

Fig. 3. CPT size comparison of a concrete example in the standard parse tree (a) and the expanded parse tree (b).

CPT size is used for a penalty term in BIC metric. Large CPT size requires a lot of samples to estimate the interactions among nodes.

In our approach, we employ the expanded parse tree for expressing individuals. In EP-I, both L and R nodes are used for selector operators. However, we use only L operator because we want to save CPT size.

3.2 Algorithm

In this section, we explain the details of proposed method POLE (Program Optimization with Linkage Estimation). A flowchart of POLE is described below which is also identical to that of other PMBGAs and PMBGPs.

Step 1 Initialization of individuals
Step 2 Evaluation of individuals
Step 3 Selection of individuals
According to the fitness values, individuals are selected which are used for construction of Bayesian network and estimation of parameters. The number of individuals to select is represented with $M \times P_s$, where P_s is a selection rate and M is a population size. Any selection methods can be employed and we use a truncate selection.
Step 4 Construction of Bayesian network
Bayesian network is constructed with the samples and parameters are also estimated. In our implementation, networks are constructed from scratch at each generation.
Step 5 Generation of new individuals
New individuals are generated with constructed Bayesian network. If we want to use the elitist strategy, better $M \times P_e$ individuals are copied from previous generation, where P_e denotes elite rate.

Until termination criteria are met, steps from 2 to 5 are repeated.

Construction of Bayesian network We adopted Bayesian network for the probabilistic model because Bayesian network is a very flexible model.

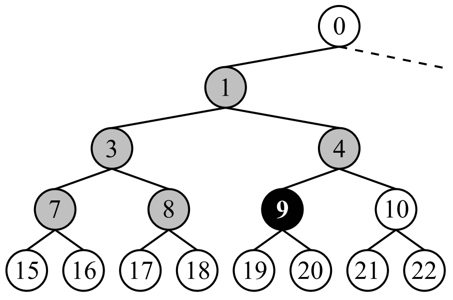

Fig. 4. The parent candidacies for a $node_9$ in a case of $R_P = 2$ are described as gray nodes.

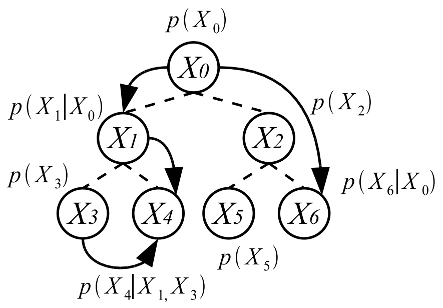

Fig. 5. Network example.

Construction of Bayesian network from given data is known to be a very computationally expensive task. Many algorithms have been proposed up to now and we employed the combination of BIC (Bayesian Information Criteria) [16] scoring metric and K2 algorithm [18]. K2 algorithm can construct Bayesian network relatively fast, because K2 algorithm make the ordering on the variables. In K2 algorithm, the parent candidacies of $node_i$ are nodes which have smaller indexes. K2 algorithm tries to add parents from parent candidacies repeatedly. The parent addition which most increases the score is applied to the network.

K2 algorithm can construct Bayesian network relatively fast. However, because GP requires a lot of nodes during search, K2 algorithm may not be fast enough for the program evolution. In the tree structure, interactions among neighbor nodes may be stronger than those among distant nodes. Our algorithm takes advantage of this assumption for a fast network construction.

In POLE, the nodes which satisfy the following two conditions are parent candidacies of $node_i$ (Nodes are numbered with breadth-first traversal), where $U(i, R_p)$ is the node R_p levels above $node_i$.

- Nodes which belong to the sub-tree rooted at $U(i, R_p)$
- Nodes whose indexes are smaller than i

Figure 4 is an example of parent candidacies. This figure shows parent candidacies of $node_9$ ($R_p = 2$). The candidacies are described with gray nodes. Figure 5 is an example of Bayesian network structure in POLE. Edges are generated among nodes to estimate the interactions among nodes.

Generation of new individuals New individuals are generated with Bayesian network and CPT. First, the nodes which have no parent nodes are decided with the probability distribution. Then nodes whose parents are already fixed are conditionally generated. Because the Bayesian network is DAG (Directed Acyclic Graph), all nodes can be decided in turn.

Table 1. Main paremeters for POLE.

Symbol	Meaning	Value
P_s	Selection rate	0.1
P_e	Elite rate	0.005
k	The number of maximum incoming edges	∞
R_p	Parent range	2
P_F	Function selection rate at the initialization	0.8

4 Comparative Experiment

In order to show the effectiveness of our approach, we applied POLE to a deceptive MAX (DMAX) problem. In the experiments, we used 4 models itemized below for comparison.

- **POLE**
  This is a proposed method. To estimate the interactions among node, Bayesian network is constructed. The number of maximum incoming edges per node is not limited. For selection, a truncate selection is used and for initialization, Grow is adopted. Parameters are listed in Table 1.
- **MODEL A**
  This algorithm is a $k = 0$ case of POLE. No network is constructed in this algorithm and no interactions among nodes are estimated. The search mechanism resembles that of PIPE. Other setups are identical to POLE.
- **MODEL B**
  This model considers parents and children relationships in the tree structure as EDP. However, this algorithm uses the expanded parse tree and this is the difference between MODEL B and the original EDP. Other setups are identical to POLE.
- **Simple GP (SGP)**
  This algorithm is a simple implementation of GP. In the experiments, $P_e = 0.005$, $P_c = 0.995$, $P_m = 0$, $P_r = 0$ are used. Crossover points are selected with bias. Crossovering the two individuals, the first crossover point is selected from functions and terminals with the possibilities of 0.9 and 0.1, respectively. The second point is selected under the condition that the depth of both individuals does not exceed the depth limitation. For selection, a tournament selection is used, and for initialization of individuals, Grow is adopted.

4.1 Deceptive MAX (DMAX) Problem

The MAX problem [19, 20] is a benchmark test for investing the search mechanism of GP, and is widely used as a benchmark test for PMBGP [9, 13]. An objective of the MAX problem is to find the function which returns the largest value under limitation on maximum tree depth. Symbols are also limited ($F = \{+, \times\}$, $T = \{0.5\}$). To investigate the search performance of POLE, we applied our approach to a deceptive MAX problem, which is an extended problem of the MAX problem. Because the original MAX problem does not have deceptiveness, it is very easy for some PMBGPs which do not consider the interactions to solve. We extended the MAX problem by adding the deceptiveness and we call this extended problem *deceptive MAX problem* (DMAX problem).

A main objective of the DMAX problem is identical to the original one: to find the functions which return the largest real value under the limitation on maximum tree depth D. However, the symbols used in the DMAX problem are different from those used in the MAX problem. The DMAX problem uses the symbols represented with Equation 2, where add_m and $multiply_m$ are m arity function and defined as Equation 3. Terminal χ is a complex value and represented in Equation 2. A fitness value of individual is a real part of its function. If a value of a function is $a+bi$ (where $i=\sqrt{-1}$), then its fitness value is a. The DMAX problem can be represented with 3 parameters, m (arity), r (root of rth power) and D (maximum tree depth).

$$F=\{add_m, multiply_m\}, \quad T=\{\chi, 0.95\}, \quad \chi=\cos\frac{2\pi}{r}+i\sin\frac{2\pi}{r} \tag{2}$$

$$add_m(a_0,...,a_{m-1})=\sum_{j=0}^{m-1}a_j, \quad multiply_m(a_0,...,a_{m-1})=\prod_{j=0}^{m-1}a_j \tag{3}$$

The difficulty of this problem depends on these three parameters. In this paper, we experimented with $m=5$, $r=3$, $D=3$ and 4 which has a very strong deceptiveness (Section 4.1).

Let us consider the optimum value (the maximum value) in the DMAX problem of $m=5$, $r=3$ and $D=3$. In this setup, $\chi^3=1$. First, add χ with add_5 to make 5χ. Then multiply this value with $multiply_5$ and the value becomes $(5\chi)^5=5^5\chi^5=5^5\chi^2$. However, $\mathrm{Re}(5^5\chi^2)$ is a negative value and is not a good solution. So 2 values out of 5 values to be multiplied have to be real values. 5×0.95 is used as a substitution for 5χ and the maximum value is $(5\chi)^3(0.95\times 5)^2=2820.3125$. In $D=4$, the maximum value is $(5\chi)^{24}(0.95\times 5)\approx 2.83\times 10^{17}$.

In the normal MAX problem, it is very easy to get the maximum value with multiplying $(0.5+0.5+0.5+0.5)$. On the other hand, DMAX is more difficult to solve because of the deceptiveness.

Experimental setups We used the DMAX problem of $m=5$, $r=3$, $D=3,\ 4$. In every algorithm, we started from $M=100$ and increase M by $\sqrt[5]{10}$ times ($M=100,160,250,400,\ 630,1000,1600...$) until the algorithm achieves the possibility of success at 100% (20 runs). The maximum population size is $M=25000$. We observed the average number of fitness evaluations in each algorithm. In SGP, tournament size is set to $T=2$.

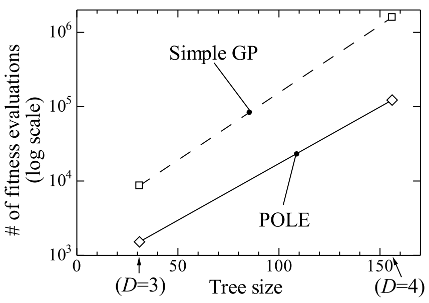

Fig. 6. The number of fitness evaluations ($m = 5$, $r = 3$). The results of MODEL A and MODEL B are omitted, because these algorithms could not achieve the possibility of success = 100% in $D = 4$ even with $M = 25000$.

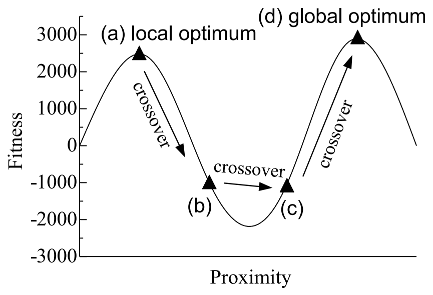

Fig. 7. An abstract image of DMAX fitness landscape when using GP. Horizontal axis stands for the structural proximity. (a) $\sim$ (d) correspond to those in Figure 8.

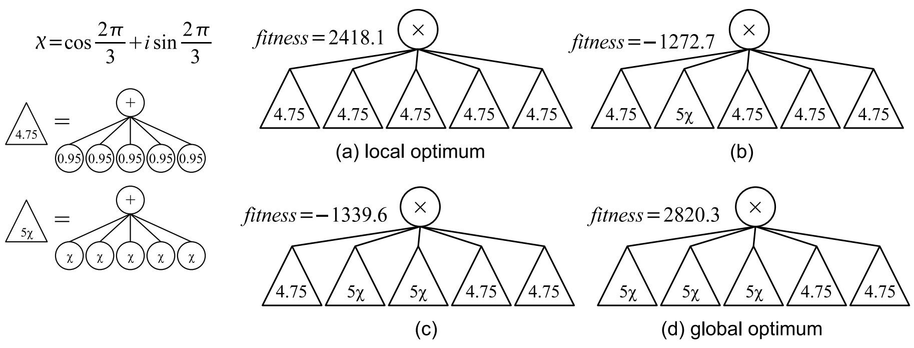

Fig. 8. The local (a) and global (d) optima in the DMAX problem ($m = 5$, $r = 3$, $D = 3$). (b) and (c) show intermediate structures.

Results and discussion Figure 6 describes the average number of fitness evaluations in 20 runs. Because MODEL A and MODEL B was not able to achieve the possibility of success = 100% in $D = 4$ even with $M = 25000$, we did not show the results of them. In this figure, the horizontal axis represents tree size $((5^D - 1)/(D-1))$. According to this figure, we can see that our approach requires much less number of fitness evaluations than SGP, not to mention, MODEL A and MODEL B. Although SGP can get the optimum solution more effectively than MODEL A and MODEL B, the number of fitness evaluations in SGP scales up more rapidly than POLE. GP can not solve the DMAX problem effectively because of deceptive fitness landscape of the problem. Figure 8(a) is a local optimum and Figure 8(d) is a global optimum of the DMAX problem ($m = 5$, $r = 3$ and $D = 3$). When an individual represented in (a) transforms to (d) with crossover, the individual have to go through the intermediate structures represented in (b) and (c). However, fitness values of (b) and (c) are negative and the possibility that these intermediate structures are selected in the next generation is very low. Figure 7 is an abstract image of DMAX fitness landscape when using GP crossover. As can been seen, (b) and (c) structures are the valleys in this fitness landscape.

MODEL A and MODEL B have more tendencies to be trapped with the local optimum than SGP, because MODEL A and MODEL B are not able to estimate the building block properly and easily destroy them. As a result, MODEL A and MODEL B are much more inferior to POLE in this problem. With the same population size as that of POLE with which POLE acheived the possibility of success = 100% in $D = 4$, MODEL A and MODEL B could not find the optimum out of 20 runs.

Because our approach estimates the interactions among node effectively, POLE can get the optimum value with the least number of fitness evaluations. In the DMAX problem, POLE is much more effective in solving the DMAX problem than prior program evolution methods.

5 Conclusion

We have proposed Bayesian network for estimation of interactions among nodes by using the expanded parse tree which dramatically reduce the size of CPT. It has been shown that in the DMAX problem, POLE shows the superior performance compared to simple GP and other probabilistic models. POLE may be effective for the problem which has strong deceptiveness in the search space. Applications of POLE to real-world problems are our future research topic.

References

1. Hasegawa, Y., Iba, H.: Optimizing Programs with Estimation of Bayesian Network. In: Proceedings of the 2006 World Congress on Computational Intelligence, Vancouver, IEEE press (2006) 5527–5534
2. Pelikan, M.: Bayesian optimization algorithm: From single level to hierarchy. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL (2002) Also IlliGAL Report No. 2002023.
3. Larrañaga, P., Lozano, J.A.: Estimation of Distribution Algorithms. Kluwer Academic Publishers (2001)
4. Sastry, K., Goldberg, D.E.: Probabilistic model building and competent genetic programming. In Riolo, R.L., Worzel, B., eds.: Genetic Programming Theory and Practise. Kluwer (2003) 205–220
5. Pelikan, M., Goldberg, D.E., Cantú-Paz, E.: BOA: The Bayesian optimization algorithm. In Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakiela, M., Smith, R.E., eds.: Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99. Volume I., Orlando, FL, Morgan Kaufmann Publishers, San Fransisco, CA (1999) 525–532

6. Etxeberria, R., Larrañaga, P.: Global optimization using Bayesian networks. Proc. 2nd Symposium on Artificial Intelligence (CIMAF-99) (1999) 332 – 339
7. Mühlenbein, H., Mahnig, T.: FDA - A scalable evolutionary algorithm for the optimization of additively decomposed functions. Evolutionary Computation **7** (1999) 353–376
8. Salustowicz, R.P., Schmidhuber, J.: Probabilistic incremental program evolution: Stochastic search through program space. In van Someren, M., Widmer, G., eds.: Machine Learning: ECML-97. Volume 1224., Springer-Verlag (1997) 213–220
9. Yanai, K., Iba, H.: Estimation of distribution programming based on Bayesian network. In Sarker, R., Reynolds, R., Abbass, H., Tan, K.C., McKay, B., Essam, D., Gedeon, T., eds.: Proceedings of the 2003 Congress on Evolutionary Computation CEC2003, Canberra, IEEE Press (2003) 1618–1625
10. Looks, M., Goertzel, B., Pennachin, C.: Learning computer programs with the Bayesian optimization algorithm. In Beyer, H.G., et al, eds.: GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation. Volume 2., Washington DC, USA, ACM Press (2005) 747–748
11. Ratle, A., Sebag, M.: Avoiding the bloat with probabilistic grammar-guided genetic programming. In Collet, P., Fonlupt, C., Hao, J.K., Lutton, E., Schoenauer, M., eds.: Artificial Evolution 5th International Conference, Evolution Artificielle, EA 2001. Volume 2310 of LNCS., Creusot, France, Springer Verlag (2001) 255–266
12. Shan, Y., McKay, R.I., Abbass, H.A., Essam, D.: Program evolution with explicit learning: a new framework for program automatic synthesis. In Sarker, R., Reynolds, R., Abbass, H., Tan, K.C., McKay, B., Essam, D., Gedeon, T., eds.: Proceedings of the 2003 Congress on Evolutionary Computation CEC2003, Canberra, IEEE Press (2003) 1639–1646
13. Shan, Y., McKay, R.I., Baxter, R., Abbass, H., Essam, D., Hoai, N.X.: Grammar model-based program evolution. In: Proceedings of the 2004 IEEE Congress on Evolutionary Computation, Portland, Oregon, IEEE Press (2004) 478–485
14. Bosman, P.A., de Jong, E.D.: Grammar transformations in an EDA for genetic programming. Technical Report UU-CS-2004-047, Institute of Information and Computing Sciences, Utrecht University (2004)
15. Regolin, E.N., Pozo, A.T.R.: Bayesian automatic programming. In Keijzer, M., Tettamanzi, A., Collet, P., van Hemert, J.I., Tomassini, M., eds.: Proceedings of the 8th European Conference on Genetic Programming. Volume 3447 of Lecture Notes in Computer Science., Lausanne, Switzerland, Springer (2005) 38–49
16. Schwarz, G.: Estimating the Dimension of a Model. The Annals of Statistics **6 (2)** (1978) 461 – 464
17. Wineberg, M., Oppacher, F.: A representation scheme to perform program induction in a canonical genetic algorithm. In Davidor, Y., Schwefel, H.P., Männer, R., eds.: Parallel Problem Solving from Nature III. Volume 866 of LNCS., Jerusalem, Springer-Verlag (1994) 292–301
18. Cooper, G., Herskovits, E.: A Bayesian method for the induction of probabilistic networks from data. Machine Learning **9** (1992) 309–347
19. Gathercole, C., Ross, P.: An adverse interaction between crossover and restricted tree depth in genetic programming. In Koza, J.R., Goldberg, D.E., Fogel, D.B., Riolo, R.L., eds.: Genetic Programming 1996: Proceedings of the First Annual Conference, Stanford University, CA, USA, MIT Press (1996) 291–296
20. Langdon, W.B., Poli, R.: An analysis of the MAX problem in genetic programming. In Koza, J.R., Deb, K., Dorigo, M., Fogel, D.B., Garzon, M., Iba, H., Riolo, R.L., eds.: Genetic Programming 1997: Proceedings of the Second Annual Conference, Stanford University, CA, USA, Morgan Kaufmann (1997) 222–230