Sampling Bias in Estimation of Distribution Algorithms for Genetic Programming using Prototype Trees

Kangil Kim, Bob (R.I.) McKay, and Dharani Punithan

Structural Compexity Laboratory, Seoul National University, Korea
`https://sc.snu.ac.kr`

Abstract. Probabilistic models are widely used in evolutionary and related algorithms. In Genetic Programming (GP), the Probabilistic Prototype Tree (PPT) is often used as a model representation. Drift due to sampling bias is a widely recognised problem, and may be serious, particularly in dependent probability models. While this has been closely studied in independent probability models, and more recently in probabilistic dependency models, it has received little attention in systems with strict dependence between probabilistic variables such as arise in PPT representation. Here, we investigate this issue, and present results suggesting that the drift effect in such models may be particularly severe – so severe as to cast doubt on their scalability. We present a preliminary analysis through a factor representation of the joint probability distribution We suggest future directions for research aiming to overcome this problem.

1 Introduction

A wide range of evolutionary algorithms learn explicit probability models, sampling individuals from them, using the fitness of individuals to update the model. They range from Colorni and Dorigo's Ant Colony Optimization (ACO) [1] and Baluja's Population Based Incremental Learning (PBIL) [2] through Muehlenbein and Manig's Factorized Distribution Algorithm (FDA) [3] or Pelikan's Bayesian Optimization Algorithm (BOA) [4] to Salustowicz and Schmidhuber's Probabilistic Incremental Program Evolution (PIPE) [5]. Historically, different strands of this research have developed in relative isolation, and there is no acknowledged single term to describe them. In this paper, we refer to such algorithms as Estimation of Distribution Algorithms (EDAs), acknowledging that this may be wider-than-normal usage.

When EDAs are applied to Genetic Programming (GP) [6] problems, the most obvious question is what statistical model to use to represent the GP solution space, and how to learn it. This question has drawn most of the attention of researchers in this field, with consequent neglect of the sampling stage of EDA-GP algorithms.

In GP, many EDAs have used variants of the Probabilistic Prototype Tree (PPT) as their proability model, beginning with PIPE [5] and extending to Yanai and Iba's EDP [7], Sastry et al.'s ECG) [8], Hasegawa and Iba's POLE [9], Looks et al.'s BOAP [10] and Roux and Fontupt's Ant Programming [11]. The PPT is a convenient model for representing probability distributions estimated from tree individuals. However Hasegawa and Iba already noted that it suffers from some representational problems, and proposed the Extended Parse Tree (EPT) variant [9]. What has not been studied is the effect on sampling drift of its implicit dependence model.

Sampling drift effect is an important problem for all probability models. However the strict probability dependence in the PPT greatly amplifies this effect relative to the other major sources of bias in EDAs (selection pressure and learning bias), thus becoming a critical issue in scaling of PPT-based EDAs to large-scale problems.

In this paper, we examine this problem both empirically and mathematically. We designed two simple problems, closely related to the well-known one-max and max problems, with simple fitness landscapes to reduce the effects of other factors. We compare the behaviour of a PIPE model with a PBIL-style independent model to illustrate the amplified effect of sampling bias. We mathematically investigate how the factorised distribution implicit in the PPT model causes this increased sampling bias.

In section 2, we present a brief overview of EDAs and of PPTs.. The experiments are described in section 3, with their results following in section 4. Section 5 analyse the factorisation implicit in the PPT. We discuss the implications of these results in section 6, drawing conclusions and proposing future directions in section 7.

2 Background Knowledge

2.1 Estimation of Distribution Algorithms

EDAs are evolutionary algorithms incorporating stochastic models. They use the key evolutionary concepts of iterated stochastic operations as shown below:

```
generate N individuals randomly
while not termination condition do
  Evaluate individuals using fitness function
  Select best individuals
  Construct stochastic model from selected individuals
  Sample new population from model distribution
end while
```

They differ from a typical evolutionary algorithm only in model construction and sampling. All EDAs use some class $\mathcal{M}$ of probability models, and a corresponding decomposition of the structure of individuals. Model construction specifies a model from $\mathcal{M}$ for each component. Sampling a new individual traverses the components, sampling a value from each model, so that the sample component distribution reflects the model's. In the simplest version, PBIL, the probability model is a vector of independent probability tables, one for each location of the phenotype.

2.2 Probabilistic PrototypeTrees and EDAs

PPT-based EDAs use a tree structure to store the probability distribution. Given a predefined instruction set of maximum arity n, the PPT is an n-ary full tree storing a probability table over the set of instructions. PPT was first used in PIPE [5], where each node contained an independent probability table. ECGP [8] extended this by modelling dependence between PPT nodes as in the Extended Compact Genetic Algorithm [12].

EDP [7] instead conditioned each node on its parent. BOAP [10] learnt Bayesian networks (BN) of dependences in the PPT, while POLE [9] learnt BNs representing dependences in an ”Extended Parse Tree”, a variant of the PPT.

2.3 Benchmark Problems

One Max is the near-trivial problem of finding a fixed-length binary string maximising the sum of all bits [13]. Its fitness landscape is smooth with no local optima. Thus it is well-suited to the PBIL independent-probability model, using a probability vector $V = [E_1, \ldots, E_n]$ over the value set $\{0, 1\}$ to represent the locations in the string.

The Max problem is a generalisation of one-max, where the goal is to find the largest-valued tree that can be constructed from a given function set I and terminal set T, in a given depth D [14]. Typically $I = \{\times, +\}$ and $T = \{0.5\}$. This appears well-suited to the ”independent” probability model of PIPE, in that each node of the PPT – in this case, a full binary tree – holds an independent probability table, giving the probability of selecting each element of $I \cup T$. The simplest case of max, $I = \{+\}, T = \{0, 1\}$ is closely related to one-max, in that once the system has found a full binary shape, the remaining problem, of filling the leaves with 1, is essentially the one-max problem. We note that in making this comparison, we are, in effect, mapping the nodes of the PPT tree to corresponding locations in a PBIL chromosome.

2.4 Grammar Guided Genetic Programming

To set the context for this study, we compare the performance of GP on the same problems; we can’t use a standard GP system for this, because it is unable to enforce the constraints of the one-max problem. For fair comparison, we use a Grammar Guided GP system (GGGP) [15].

3 Experimental Analysis

Our experiments illuminate sampling drift in PPT-based EDAs, comparing it with a well-understood model (PBIL). We need to specify four aspects:

1. the probability model structures
2. the fitness functions
3. the EDA algorithm
4. experimental parameters

To illustrate, we use the max problem, and a slight variant of one-max, with the same target as max (but a more one-max-like fitness function). We compare with a conventional GGGP approach to show the intrinsic simplicity of these problems. For economy of explanation, we describe the max problem first.

3.1 Model Structures

The genotype representation is a 15-long string $X = X_1, \ldots, X_{15}$. This can be used in either of two ways: the string can be modelled through an independent, PBIL-style genotype, or it can be mapped to a binary PPT of depth 3 (which has 15 nodes).

In the PBIL structure each location contains an independent probability table with three possible values, $+$, $\times$ and 0.5. The table is used to generate sample values at each generation, then is updated to reflect the sample distribution of the selected individuals.

In the PPT structure each location contains an independent probability table over the values $+$, $\times$ and 0.5, but each (except the leaves) has two children, with the relationship:

$$\begin{aligned} \text{left child}\,(X_i) &= X_{i\times 2} \\ \text{righ child}\,(X_i) &= X_{i\times 2+1} \end{aligned}$$

"Independence" in the latter case must be taken with a grain of salt. While the probability tables in the PBIL structure are independent, the PPT structure introduces a dependence: the descendants of a node holding the (terminal) value 0.5 are not sampled. This is the primary issue under consideration here.

3.2 Max Problem Fitness Function

Fitness is defined by the following equation:

$$\text{itFit}\,(X_i) = \begin{cases} \text{itFit}\,(\text{left child}\,(X_i)) \times \text{itFit}(\text{right child}\,(X_i)) & \text{if} X_i = \times, 1 \le i \le 7 \\ \text{itFit}\,(\text{left child}\,(X_i)) + \text{itFit}\,(\text{right child}\,(X_i)) & \text{if} X_i = +, 1 \le i \le 7 \\ 0.0 & \text{if} X_i = \times, 8 \le i \le 15 \\ 0.0 & \text{if} X_i = +, 8 \le i \le 15 \\ 0.5 & \text{if} X_i = 0.5 \end{cases}$$

When $+$, $\times$ were used in leaf nodes, there is a problem in allocating fitness, since they have no children. To overcome this, in this case we give them fitness 0. The maximum value of this function (the target) corresponds to a full binary tree with $+$ in the bottom two layers, and $+$ or $\times$ in the top layer.

3.3 Variant One-max Problem Fitness Function

The task is to find a string having a specific value in each location, defined by dividing the locations into three groups, as in equations 1.

$$\begin{aligned} L_1 &= \{X_1\} \\ L_2 &= \{X_i\}\ 2 \le i \le 7 \\ L_3 &= \{X_i\}\ 8 \le i \le 15 \end{aligned} \tag{1}$$

In this case, the fitness function is given by equation 2:

$$\text{omFit}\,(X) = \textstyle\sum_{i=1}^{15}\ \text{locFit}\,(X_i) \tag{2}$$

where

$$\text{locFit}\left(X_i\right) = \begin{cases} 1 \;\text{ if } X_i = \times & \text{and } X_i \in L_1 \\ 1 \;\text{ if } X_i = + & \text{and } X_i \in L_2 \\ 1 \;\text{ if } X_i = 0.5 & \text{and } X_i \in L_3 \\ 0 \;\text{ else} \end{cases}$$

This differs from the typical one-max problem in two ways: there are three possible values, not two, and target values at differ with location. However neither makes much difference to the fitness landscape, which remains smooth, with no local optima.

3.4 EDA System

In these comparisons, we use a very simple EDA system so that the implications of the experiments are clear. In detail:

Selection: truncation. Given a selection ratio λ, the top λ proportion of individuals are selected. We varied the selection ratio λ to investigate the effect and scale of drift.

Model Update: the model structure was fixed for the whole evolution. Maximum likelihood was used to estimate the probabilities from the selected sample.

Sampling: we used Probabilistic Logic Sampling [16], the most straightforward sampling method, used in most EDA-GP systems.

To simplify understanding, two common EDA mechanisms which can slow drift, elitism and mutation, were omitted from the system

3.5 Parameter Settings

We used truncation selection with selection ratios ranging from 10% to 100% at a 10% interval. The population size was 100, and the algorithm was run for 200 generations. Each setting was run 30 times. Detailed parameters settings for the GGGP and EDA-GP runs are shown in table 1, while the grammar used for GGGP (with starting symbol EXP$_1$) is shown in table 2

Table 1. Experimental Parameter Settings

General Parameters	**Value**	**EDA Parameters**	**Value**	**GGGP Parameters**	**Value**
Genotype		Operators		Operators	
Length	15	Selection	Truncation	Selection	Tournament
Values	$+, \times, 0.5$	Ratios	$0.1, \ldots, 1.0$	Size	5
		Update	Max. Likelihood	Cross. prob.	0.5
		Sampling	PLS	Mut. prob.	0.75
Population	50	Dependence		Reproduction	Generational
Generations	200	PBIL	independent		
Runs	30	PPT	PPT		

Table 2. GGGP Grammar

$\mathrm{EXP}_i \rightarrow \mathrm{EXP}_{i+1}$ OP EXP_{i+1}		$(0 < i < 4)$
$\mathrm{EXP}_4 \rightarrow$ OP		
OP $\rightarrow +\vert\times\vert 0.5$		

4 Result of Preliminary Experiments

4.1 One-Max Results

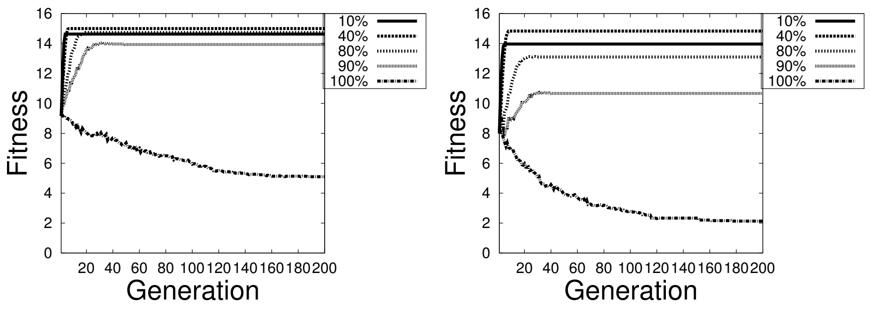


Fig. 1. Best Fitness vs Generation for One-max Variant (Structure : left, $PBIL$, right, PPT; percentage is the selection ratio)

Figure 1 shows the performance of the two probability models, at various levels of selection. Each plot shows a particular structure for a range of different selection ratios. Each line represents the best fitness achieved in each generation, for a particular selection ratio. By comparison, GGGP finds perfect solutions in 14.3 ± 4.9 generations.

We note that even for this near-trivial fitness function, PPT shows worse performance than PBIL. In the left-hand plot, the PBIL structure finds a solution close to the optimum (15) at most selection ratios other than 90% and 100% (i.e. no selection). These results are replicated for the selection ratios not plotted, most showing performance very close to the optimum, as with the 40% selection ratio. By comparison, the PPT model shows much worse performance. In all selection ratios, PPT converges to sub-optimal solutions. The difference increases with weaker selection, with the 100% ratio showing a substantial decrease in fitness, below that achieved by random sampling. With selection pressure turned off, this drift is the result purely of sampling. With increasing selectivity, the drift effect becomes weaker, but still acts counter to the selection pressure.

4.2 Max Problem

This problem is much tougher than the previous. GGGP finds perfect solutions in 17.8 ± 8.0 generations. However EDA performance fares far worse. The PBIL model is unable

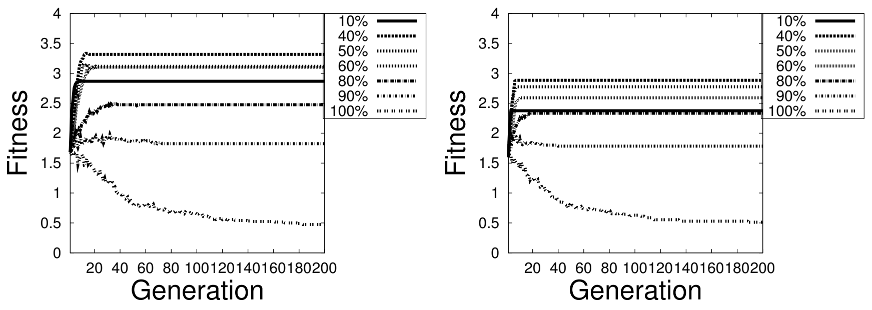


Fig. 2. Best Fitness vs Generation for Max (Structure : left, $PBIL$, right, PPT, percentage is the selection ratio)

to find the optimum solution (4) at any selection ratio, and the differences from the optimum are larger than for one-max. Given that the fitness function has epistasis, which PBIL is unable to model, this is not surprising. What is surprising is the even poorer performance of the PPT model. PPT appears well-matched to the fitness function, yet performs much worse than the naive PBIL model. PBIL is able to achieve fitnesses, for some selection ratios, of around 3.4, whereas PPT never exceeds 2.7. the effects are particularly marked around selection ratios from 10% through to 60%, with the differences becoming weaker by 80% to 90%, and essentially disappearing at a 100% selection ratio.

4.3 Performance of PPT

Overall, we see poor performance from the PPT model for both simple and complex problems. Even for the max problem – the kind of problem that PPT was designed to solve – it shows much worse performance than PBIL. The behaviour under 100% selection – i.e. pure sampling drift – suggests a possible cause: that sampling drift [17] may be the major influence on peformance. The poor performance on the trivial fitness landscape of the one-max variant supports this. The good performance of GGGP emphasizes just how damaging this effect is.

5 Analysis of the PPT Model

5.1 The Effects of Arity

In a PPT, each node represents a random variable which can take any of the possible instructions as its value.[1] Table 3 shows a typical example for the case of symbolic regression, with a function set consisting of the four binary arithmetic operators, four unary trigonometric and exponential operators, and a variable and constant, of arity 0.

[1] nodes at the maximum depth are only permitted values of zero arity, but for the sake of simplicity we omit this from consideration here.

Table 3. PPT Table for Symbolic Regression, Showing Arities

Instruction	Arity	Probability	Instruction	Arity	Probability
+	2	0.1	sin	1	0.1
$\times$	2	0.1	cos	1	0.1
-	2	0.1	log	1	0.1
/	2	0.1	exp	1	0.1
x	0	0.1	C	0	0.1

The combining of nodes of different arities in the PPT model creates a dependence relationship between parent and child nodes, even though their probability distributions appear to be separate. If a node n_1 is sampled as sin, one of the child nodes – conventionally n_3 – loses the opportunity to sample an instruction. Therefore the probability of sampling n_3 is different from that of n_2, the other child node. Thus although the probability distribution of n_3 is independent of the condition set of n_1, n_3 is nevertheless dependent on the complete condition set of n_1, because the probability of sampling an instruction for n_3 is 0 in the case where a unary function or variable is sampled at n_1.

To clarify this dependency, we transform the PPT probability distribution to a semi-degenerate Bayesian network.[2]

5.2 Conversion to Semi-Degenerate Bayesian Network

Undefined instruction In the PPT, each node's probability table cannot be directly treated as a random variable, because the probability distribution for some conditions of the parent is not recorded in the table. To cover this case, where a node can not select any value, we define an additional value U, for 'undefined value'. Taking a simple case with just three values, $+$, sin and C, an independent PPT might have probabilities of 0.4 for $+$ and sin, and 0.2 for C. Taking account of the parent-child dependencies, we could represent the overall conditional dependency of a random variable for a node given its parent, as in figure 3. In the parent node of $M4$, any of $+$, sin, C or U might be sampled. When C, constant, is sampled, $M4$ is not able to sample any value, so that the probabilities for selecting $+$, sin and C are zero; to represent that no instruction can be sampled in this condition, we allocate the 'undefined' instruction a probability of 1.0. If the parent node is sampled as 'undefined', $M4$ must also be undefined.

Figure 4 shows more detail, illustrating how a simple three-node PPT can be transformed into a (semi-degenerate) BN. Note that the probability structures of the left and right children differ (because of the differing effects of the sin function in the parent).

5.3 Factorization of Full Joint Distribution

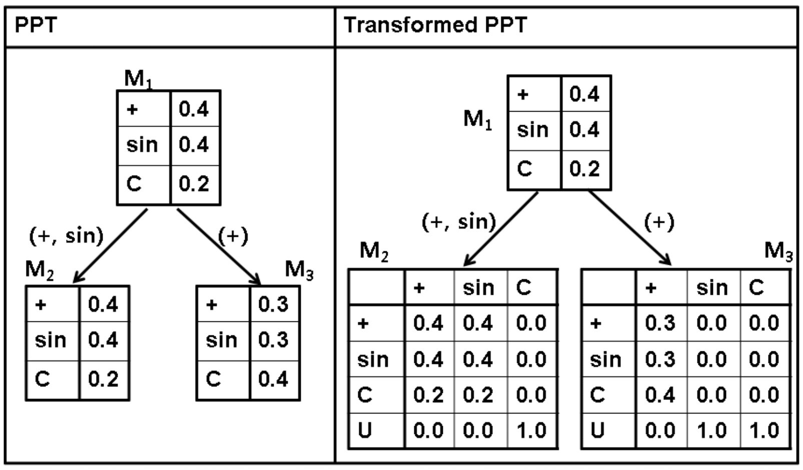
Dependent variable In the resulting BN, the transformed nodes become conditionally dependent on their parent nodes (there are only two exceptions – either the node is

[2] In standard terminology, tables without zeros are said to be non-degenerate, and tables containing only 0.0 and 1.0 are degenerate. We introduce the term 'semi-degenerate' for the intermediate case, of tables containing 0.0 but not necessarily 1.0.

M_4

	+	sin	C	U
+	0.4	0.4	0.0	0.0
sin	0.4	0.4	0.0	0.0
C	0.2	0.2	0.0	0.0
U	0.0	0.0	1.0	1.0

Fig. 3. Transformed Probability table of PPT



PPT

M_1

+	0.4
sin	0.4
C	0.2

M_2

+	0.4
sin	0.4
C	0.2

M_3

+	0.3
sin	0.3
C	0.4

Transformed PPT

M_1

+	0.4
sin	0.4
C	0.2

M_2

	+	sin	C
+	0.4	0.4	0.0
sin	0.4	0.4	0.0
C	0.2	0.2	0.0
U	0.0	0.0	1.0

M_3

	+	sin	C
+	0.3	0.0	0.0
sin	0.3	0.0	0.0
C	0.4	0.0	0.0
U	0.0	1.0	1.0

Fig. 4. Transformation from PPT to semi-degenerate BN

always undefined, hence unreachable and may be omitted from the PPT, or else the node is always defined, implying that the parent node cannot sample a terminal, an unreasonable situation in GP – both may be safely ignored).

In the simplest PPT case, where each node's value is assumed probabilistically independent of the other nodes, the only dependence is that arising above. That is, this simple case corresponds to the assumption that each node is conditionally independent of all other nodes in the PPT, conditioned only on its parents. Thus the probability distribution of node x can be represented by $p(x|\text{parent of } x)$, and the full joint probability distribution of the transformed PPT as:

$$p(X) = \prod_i p(x_i | x_{\text{parent of } i}) \quad (3)$$

Of course, more complex dependencies between PPT nodes may give rise to more complex dependencies in the corresponding BN, but the dependence of the child on its parents will always remain.

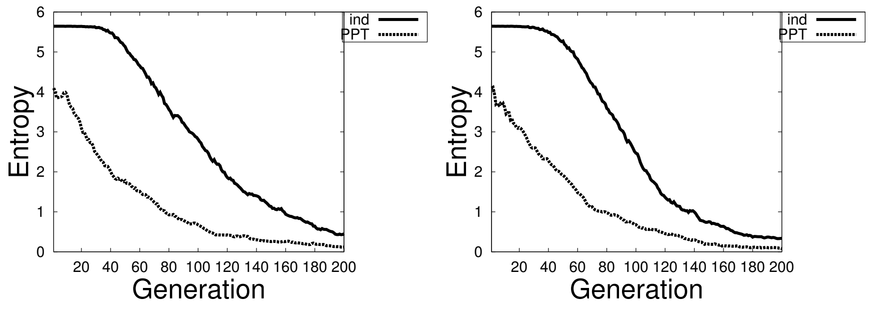


Fig. 5. Entropy of Population vs Generation (Left: One-max Variant; Right: Max (ind : independent – PBIL – structure))

Sampling bias This factorization of the joint distribution gives us a way of understanding the rapid diversity loss in PPT-based EDAs. In PLS sampling, for each random variable, the sample size is the same in the transformed PPT. However the actually meaningful instructions exclude undefined instructions. The size of the sample actually used to generate meaningful instructions reduces (exponentially) with depth. This is the cause of the rapid diversity loss due to sampling drift: unlike other EDAs, in which the sample size is the same across all variables, drift increases due to reduced sample size with depth. Figure 5, shows the population (phenotype) entropy at each generation. We only show the 100% selection ratio, because there, there is no diversity loss due to selection, the whole loss is the result of sampling drift. In both problems, the loss of diversity due to sampling drift is much greater in the PPT representation than in the PBIL representation.

6 Discussion

The importance of these results lie not merely in their direct implications for this trivial problem, but in their implications for PPT-based EDAs for GP. Compare these problems with typical GP problems. The dependency depth is atypically small, corresponding to a GP tree depth bound of only 3. The dependency branching is typical, or even slightly below average, for GP. And of course, the fitness landscape is vastly simpler than most GP problem domains. If this is so, why has EDA-GP been able to succeed, and even demonstrate good performance on some typical GP problems? We believe it is due to masking of the problem of accelerated drift under sampling in typical implementations.

These implementations generally incorporate mechanisms reducing the effect of sampling drift: better selection strategies and model update mechanisms, adding elitism and mutation all contribute to this reduction. In addition, our problem is tougher than typical GP problems in one respect: there is only one solution (two for the max problem). Most problem domains explored by GP have symmetries, so that eliminating a solution may not stymie exploration. Thus EDA-GP has been able to work well for GP test problems. However the drift effect worsens exponentially with tree depth, while these ameliorating mechanisms only scale linearly. Perhaps this is why EDA-GP has so far been limited to demonstrations on test problems rather than practical applications.

Some previous PPT research, notably Hasegawa and Iba's POLE [9], incorporates measures to ameliorate sampling drift using the Extended Parse Tree. Here, our focus is to clarify the effect of accelerated drift due to PPT dependency, as a preliminary to investigating solutions.

7 Conclusions

Diversity loss due to sampling is a well-known problem in EDA research, and has been carefully studied for independent probability models. It is well-known that the the problem worsens in probabilistic dependency models, and some lower bounds for the effect have already been found [17]. However there does not appear to have been previous publication of the effects on PPT-based (branching) EDAs.

By studying the sampling drift effect of two structures, on a near-trivial optimisation problem and another only slightly harder, we were able to see the importance of this diversity loss. The effects are sufficient to cast doubt on the scalability of most current approaches to EDA-GP. Can these problems be overcome? Can scalable EDA-GP systems be built? We believe it to be possible, but not easy. Any remedy must counteract the depth dependence of the drift. This probably eliminates variants of some of the traditional methods. For example, it is difficult to see how to incorporate dependence depth into population-based mechanisms such as elitism. Similarly, it doesn't seem easy to use mutation or similar mechanisms in a useful depth-dependent way. On the other hand, it may be possible to incorporate depth-based mechanisms into model update and/or sampling in ways that might be able to overcome the depth-dependence of sampling drift, and so permit scaling.

In the near future, we plan to extend this work in three directions. The first, already in progress, involves experimental measurement of diversity loss to gauge the extent of acceleration of the sampling drift effect. The second, in prospect, will attempt to mathematically estimate the diversity loss through sample size estimation. The third extends this work to grammar-based GP EDA systems (i.e. those not based on PPTs). Similar problems of accelerated sampling bias occur in these systems, though it is more difficult to isolate clear demonstrations of this.

Acknowledgment

We would like to thank Pedro Larranãga, Concha Bielza and Xuan Hoai Nguyen for their insightful involvement in discussions that led to this work. This work was supported by the Brain Korea 21 Project. The ICT at Seoul National University provided research facilities for this study.

References

1. Colorni, A., Dorigo, M., Maniezzo, V., et al.: Distributed Optimization by Ant Colonies. In Varela, F.J., Bourgine, P., eds.: Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life, MIT Press (1991) 134–142

2. Baluja, S.: Population-based incremental learning: A method for integrating genetic searching based function optimization. Technical Report CMU-CS-94-163, Computer Science Dept, Carnegie Mellon University, Pittsburgh, PA, USA (1994)
3. Mühlenbein, H., Mahnig, T.: The factorized distribution algorithm for additively decomposed functions. In: Proceedings of the 1999 Congress on Evolutionary Computation, IEEE Press (1999) 752–759
4. Pelikan, M., Goldberg, D., Cantu-Paz, E.: BOA: The Bayesian optimization algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99. Volume 1. (1999) 525–532
5. Salustowicz, R., Schmidhuber, J.: Probabilistic incremental program evolution. Evolutionary Computation **5**(2) (1997) 123–141
6. Koza, J.: Genetic programming: on the programming of computers by means of natural selection. The MIT press (1992)
7. Yanai, K., Iba, H.: Estimation of distribution programming based on bayesian network. In: Proceedings of the Congress on Evolutionary Computation, Canberra, Australia (Dec 2003) 1618–1625
8. 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
9. Hasegawa, Y., Iba, H.: A bayesian network approach to program generation. IEEE Transactions on Evolutionary Computation **12**(6) (2008) 750–764
10. Looks, M., Goertzel, B., Pennachin, C.: Learning computer programs with the bayesian optimization algorithm. In: GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation, New York, NY, USA, ACM (2005) 747–748
11. Roux, O., Fonlupt, C.: Ant programming: or how to use ants for automatic programming. In: Proceedings of the Second International Conference on Ant Algorithms (ANTS2000), Belgium (2000)
12. Harik, G., Lobo, F., Sastry, K.: Linkage Learning via Probabilistic Modeling in the Extended Compact Genetic Algorithm (ECGA). In: Scalable Optimization via Probabilistic Modeling. Volume 33. Springer (2006) 39–61
13. Schaffer, J., Eshelman, L., Offutt, D.: Spurious correlations and premature convergence in genetic algorithms. Foundations of genetic algorithms (1991) 102–112
14. Gathercole, C., Ross, P.: An adverse interaction between crossover and restricted tree depth in genetic programming. In: GECCO '96: Proceedings of the First Annual Conference on Genetic Programming, Cambridge, MA, USA, MIT Press (1996) 291–296
15. Whigham, P.A.: Grammatically-based genetic programming. In Rosca, J., ed.: Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications. (1995) 33–41
16. Henrion, M.: Propagating uncertainty in bayesian networks by probabilistic logic sampling. In: Uncertainty in Artificial Intelligence 2 (UAI-86), Amsterdam, NL, North Holland (1988) 149–163
17. Shapiro, J.L.: Diversity loss in general estimation of distribution algorithms. In: Parallel Problem Solving from Nature. Volume 4193 of Lecture Notes in Computer Science., Springer (2006) 92–101