# Grammar Model-based Program Evolution

*Abstract*— **In Evolutionary Computation, genetic operators, such as, mutation and crossover, are employed to variate the individuals to generate next population. However, these fixed, problem independent genetic operators may destroy the sub-solution, usually called building blocks, instead of discovering and preserving them. One way to overcome this problem is to build a model based on the good individuals and sample this model to obtain the next population. There some research Because of the complexity of Genetic Programming (GP) tree representation, little work of this kind has been done in GP. In this paper, we propose a new method, Grammar Model-based Program Evolution (GMPE) to evolved GP program. We replace common GP genetic operator with a Probabilistic Context-free Grammar (SCFG). In each generation, a SCFG is learned and new population is generated by sampling this SCFG model. On two benchmark problem we have studied, GMPE significantly outperforms conventional GP, usually a few times faster and more reliable.**

## I. Introduction and Related Work

In Evolutionary Computation, genetic operators, such as, mutation and crossover, are employed to variate the individuals to generate next population. However, these fixed, problem independent genetic operators may destroy the sub-solution, usually called building blocks, instead of discovering and preserving them. One way of overcome this problem, is to build a model based on the good individuals and sample this model to obtain the next population. Recently, this kind of research, research on EC guided by inductively learned model, such as Estimation of Distribution Algorithm (EDA) [10], Probabilistic Model-building Genetic Algorithm (PMBGA) [15], has drawn increasing interests.

There are several reasons that lead to this increasing interests. The first reason is the theoretical attraction. The highly complex and dynamic consequences of genetic operator is extremely hard to understand and predicate. Replacing genetic operator and population with well-formed model makes it possible to understand EC. In some simple cases, EC guided by inductively learned model is a quite accurate approximation of conventional EC [7], [12]. Secondly, in terms of practical usefulness, the empirical studies have showed the superior performance of this kind of new methods, in general.

Because of the abandon of genetic operators, either partially or completely, this kind of method becomes very different from the method of conventional evolutionary computation. However, representation of individual is consistent with conventional Genetic Algorithm (GA) or Genetic Programming (GP), i.e. GA style linear string representation or GP style hierarchical tree. Although more flexible, tree representation is more complicated than linear representation. Therefore, most of current research are focus on linear representation.



In this paper, we propose a new model for program evolution, Grammar Model-based Program Evolution (GMPE). We use GP style tree representation, with no conventional genetic operator. The stochastic grammar model is learned from the superior individuals and the new population is generated by sampling this grammar. Grammar model can represent the common structure of the superior individuals (building blocks) very well. Due to flexibility of grammar model, GMPE, the method we propose, far outperform on two benchmark problems we have studied.

This paper is organized as follows. We briefly review the related work with GA style linear representation and then with GP style tree representation in this section followed by a overview of our GMPE to give the audience a flavor of our method. In Section II, we present our work in detail, including high level algorithm, the probabilistic model (Stochastic Context-free Grammar), modeling learning method. To prove the our idea, two experiments are reported in Section III. We discuss the difficult of evolving program with tree representation, comparing with evolving solution with linear representation, such as, EDA [10], and how our GMPE addresses these difficulties in Section IV. The last section is the conclusion.

### A. Related Works with GA Style Linear Representation

There is extensive similar work in evolving linear solution guided by a inductively learned model, they differ mainly in the models they choose. Among them, we perceive the following two streams.

The first is the Learnable Evolution Model (LEM) [11]. The central engine of evolution in LEM is a machine learning mode, which creates new populations by employing hypotheses about high fitness individuals found in past populations. The machine learning mode seeks reasons why certain individuals in a population are superior to others, thus to form inductive hypotheses which explicitly characterize good individuals.

The other one is Estimation of Distribution Algorithm (EDA) [13], [17]. EDA uses a probabilistic model of promising solutions to guide further exploration of the search space. EDA is a cluster of methods, ranging from methods that assume the genes in the chromosome are independent [3], [7], [12], through others that take into account pairwise interactions [4], [2], [18], to methods that can accurately model even a very complex problem structure with highly overlapping multivariate building blocks [14], [6], [16], [5].

### B. Related Works with GP Style Tree Representation

Because of the complexity of GP style tree representation, the amount of work of evolving program with tree representation is not comparable with the work dealing with linear representation. We roughly classify them into two kinds.

The first one is probabilistic model based method. It has strong connection with EDA, i.e. evolving linear solution with the guidance of probabilistic model. It includes mainly following three projects.

Probabilistic Incremental Program Evolution (PIPE) [22] combines probability vector coding of program instructions, Population-Based Incremental Learning [3], and tree-coded programs. PIPE iteratively generates successive populations of functional programs according to an adaptive probability distribution, represented as a Probabilistic Prototype Tree (PPT), over all possible programs. Each iteration uses the best program to refine the distribution. Thus, the structures of promising individuals are learned and encoded in PPT.

Extended Compact Genetic Programmming (ECGP) [23] is a direct application of ECGA [6] in tree representation. Marginal product models (MPMs) are used to model the population of genetic programming. MPMs are formed as a production of marginal distribution on a partition of the tree. ECGP decomposes or partitions the prototype tree into subtrees and builds the probabilistic models for each subtree. Apparently, the subtrees are taken as independent probabilistic variables.

Estimation of Distribution Programming (EDP) [30] tries to model the dependency of adjacent nodes in GP tree. Although there is a few possible dependencies among the adjacent nodes, only the conditional probability of child node given parent node is considered in this research.

The second kind is grammar model based method. Although stochastic grammar model is probabilistic model, it is usually presented in very specialized literature and it has quite different learning method. Therefore, for sake of clarity, we classify it into another category. Grammar, which is widely used to model the internal hierarchical structure sentences, is one of ideal formalisms for modeling GP style tree structure. The grammar model based method includes Whigham's very early work [29], ant-TAG [1], Program Evolution with Explicit Learning (PEEL) [24].

This second kind of work has some connection with Grammar Guided Genetic Programming (GGGP), i.e. using grammar to constrain search space. The individual GP tree in GGGP must respect the grammar. This overcomes the closure problem in GP and provides a more formalized mechanism for typing (cf. strongly-typed genetic programming). Actually, the grammar model can do more than just constrain the search space. Whigham's work [29], apart from normal GGGP, grammar is slightly modified during the search. The updated grammar represent the cumulated knowledge found in the process of search.

The other works, such as ant-TAG and PEEL, are radically different from normal GP and more like EDA style evolution. In both of these work, grammars are keeping updated based on the superior individuals. New generation is obtained by sampling grammars. Conventional GP genetic operators is either entirely discarded or taken as a background operators. In ant-TAG, the structure of the grammar is fixed. The probabilities attached to the grammar are updated. However, fixed grammar means that the search space can only be explored at fixed granularity. PEEL addresses this issue by allowing the change of grammar structure.

### C. Overview

In this paper, we propose a new method, Grammar Model-based Program Evolution (GMPE) to evolved GP style program. This work is similar to above mentioned ant-TAG and PEEL, in the sense that grammar model is used to model the population. However, the grammar model and its learning method are very different from them.

We replace common GP genetic operator with a Probabilistic Context-free Grammar (SCFG). In each generation, a SCFG is learned and then new population is generated by sampling this SCFG model. Technically, the SCFG is expressive enough to represent the regularity in and among individuals. GMPE employs flexible SCFG model, which makes it more generally applicable than other methods, such as ant-TAG and PEEL.

On two benchmark problem we have studied, GMPE significantly outperforms conventional GP, usually a few times faster and more reliable.

## II. Method

### A. Algorithm

Same as conventional EC, our GMPE starts with a randomly generated population. A stochastic grammar model is learned from the selected individuals of this population. The new generation is generation by sampling the learned model. Then next iteration will start again from this new population. The high level algorithm is illustrated as a flow char in Fig. 1.

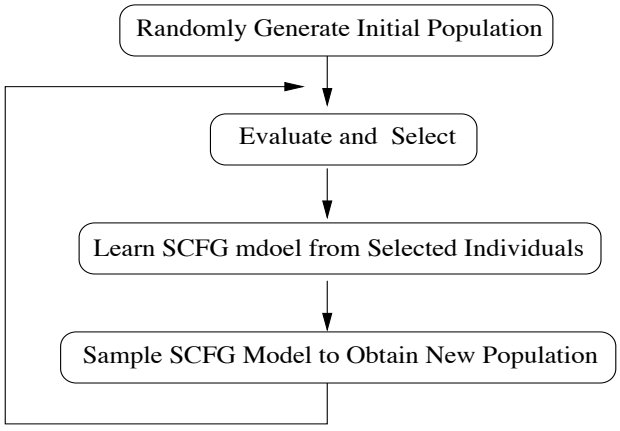


Fig. 1. Flow chart of GMPE

Since this high level algorithm is consistent with other EDA methods, we won't give more discussion on this. In the next subsection, we would mainly focus on the stochastic grammar model, which plays a critical role in GMPE.

### B. Model Learning

In this research, Stochastic Context-free Grammar (SCFG) are chosen as the model.

Actually, in GP, grammar, in particular Context-free Grammar (CFG), has been used to constrain search space [21], [28]. However, since grammar is a formal model for language, both natural and formal language, it can be more than just the constraint of search space. If we take the GP individual as string/sentence, it is not hard to see that population evolved, corresponding to corpus in Natural Language Processing (NLP), can be modeled by the grammar as well.

This subsection is organized as follows. In II-B.1, we would elaborate on the definition of SCFG and how to sample SCFG to generation the individuals. In II-B.2, the problem of learning grammar given a set of superior individuals as training samples is discussed. The learning method in this work is simple hill-climbing search. When searching for grammars, we need a measure to compare grammars. We derives this criteria based on minimal encoding inference. This measure is presented in II-B.3.

*1) Stochastic Context-free Grammar:* A stochastic contextfree grammar (SCFG) $M$ consists of

- a set of nonterminal symbols $N$ ,
- a set of terminal symbols (or alphabet) $\Sigma$,
- a start nonterminal $S \in N$ ,
- a set of productions or rules $R$,
- production probabilities $P(r)$ for all $r \in R$.
- The productions are of the form

$$X \to \lambda$$

where $X \in N$ and $\lambda \in (N \cup \Sigma)^*$ . $X$ is called the lefthand side (LHS) of the production, whereas $\lambda$ is the righthand side (RHS).

If $X \to \lambda$ is a production of $R$, the for any strings $\gamma$ and $\delta$ in $(N \cup \Sigma)^*$, we define $\gamma S \delta \Rightarrow \gamma\alpha\delta$ and we say that $\gamma S \delta$ directly derives $\gamma\alpha\delta$ in $M$. We say $\beta$ can be derived from $\alpha$, denoted $\alpha \stackrel{*}{\Rightarrow} \beta$, if there exists a sequence of direct derivations $\alpha_0 \Rightarrow \alpha_1, \alpha_1 \Rightarrow \alpha_2, \ldots, \alpha_{n-1} \Rightarrow \alpha_n$ where $\alpha_0 = \alpha, \alpha_n = \beta, \alpha_i \in (N \cup \Sigma)^*$, and $n \geq 0$. Such a sequence is call a derivation. Thus a derivation corresponds to an order of applying productions to generate a string. The probability of a derivation is the production of the probabilities of all the production rules involved. Sampling a SCFG grammar in this paper means deriving a set of strings from the given SCFG grammar. A LHS may have more than one RHSs. When deriving string, if we come to this kind of LHS, we need to choose one of its RHSs. SCFG has probability component attached to each rule (more accurately to each RHS). We need to choose the RHS based on its probability.

*2) Learning method:* We use a specific-to-general method to search for good grammar, motivated by [25]. We start from a very specialized grammar which covers only the training examples (selected superior individuals). Then *merge* operator is employed among the rules to generalize the initial grammar. Merger operator take two rules and unify its LHS with one symbol. For example, given two rules

$$X_1 \to \lambda_1$$
$$X_2 \to \lambda_2$$

after merge

$$Y \to \lambda_1$$
$$Y \to \lambda_2$$

All the other occurrence of $X_1$ and $X_2$ in the grammar should be replaced by $Y$. The reason that the merge operator generalized grammar is that before merge two rules have its own RHS, after merge two rules share these two RHSs, i.e. the merged grammar can cover more strings.

The search strategy in this research is simply hill-climbing, i.e. we randomly merge two rules in the grammar, if the score of grammar improves, this merge is accepted; we keep merging until no improvement can be found. The scoring method will be presented in the next section.

*3) Scoring:* During search, we need to compare grammars. In this case, we use hill-climbing. Therefore, we need to measure whether specific merge improves grammar model or not. We use minimum length encoding inference, usually referred to as Minimum Message Length (MML) [26], [27] or Minimum Description Length (MDL) [20] to measure the superiority of the grammar. In the remaining part of this paper, MML is employed to generally refer to minimum encoding inference.

We want to find a grammar which has low complexity but can cover training samples well. MML gives a theoretically sound framework to balance these two factors. We want to find a model (grammar in this case) to minimize the cost of coding the given data. The cost of coding the given data is the sum of the cost of coding the model and the cost of coding the data with the help of the model. Formally, we want to minimize

$$L(D) = L(G) + L(D|G) \tag{1}$$

where $D$ is the data (the corpus), $G$ is the grammar and $L(X)$ is the cost of coding $X$.

Eq. 1 is very intuitive.The two-part message states the cost of coding SCFG in the first part $L(G)$ and then the cost of coding data (training samples) given model (grammar) in the second part $L(D|G)$.

The message length of $L(D|G)$ is negative logarithm product of the probabilities of training samples (selection individuals). Each individual has a derivation. The probability of the individual is the probability of its derivation. The probability of entire set of selected individuals (training samples) is the production of probabilities of all individuals.

$L(G)$ requires the statement of the inferred SCFG. Although there are some rough estimations of $L(G)$ in the literature, they are not adequate for our purpose. We have to derived our calculation. To state a SCFG, we need to state its names of

terminal symbols, number of terminals, number of nonterminals, the RHS of each rule, the probabilities of each rule and which RHS correspond to which LHS. Formally, we have

$$\begin{aligned} L(G) &= L(\text{names of terminal symbols}) && \text{(2a)}\\ &+ L(N) + L(\Sigma) && \text{(2b)}\\ &+ L(\text{grouping of LHS}) && \text{(2c)}\\ &+ L(\text{RHS of production rule}) && \text{(2d)}\\ &+ L(\text{prob. of RHS for each nonterminal}) && \text{(2e)} \end{aligned}$$

where:

- The first term $L(\text{names of terminal symbols})$ (Eq. 2a) is the length of coding names of terminal symbols. This will be ignored because the cost is the same regardless of how many merge applied.
- The terms $L(N)$ and $L(\Sigma)$ (Eq. 2b) are the cost of coding The number of terminals and non-terminals, respectively. They can be calculated using Rissanen's methods of coding integer [20]. The length of coding integer $N$ is $\log^*(N) = \log N + \log\log N + \log\log\log N + ....$ Only positive terms are included.
- Since we do not code the LHS of the rule, we need to state which LHSs correspond to the same RHS. Let $P$ be the total number of production rules ($P \geq N$). The term $L(\text{grouping of LHS})$ should be $L(P)+L(partition(P))$, $partition(P)$ is number of possible partitions of integer $P$.
- Eq. 2d is the length of coding the RHS of each rule. The total number of distinct symbol is $N + \Sigma$. Using fixed coding scheme, each of them needs $\log(N + \Sigma)$ bits to code. Suppose the total number occurrences of symbols on RHSs is $m$, the total cost of coding them is $(m + 1)\log(N + \Sigma)$.
- The last term Eq. 2e is probabilities of production rules. Some LHS has more than one RHS. When deriving individuals, we need to know at what probability we replace this LHS with which RHS. Hence we need to record this probability. More precisely, we record the frequencies which will be then normalized to obtain the probabilities. We prefer very skewed probabilities distribution to uniform distribution because this means we would have less uncertainty. This preference can be modeled by a symmetric Dirichlet prior.

$$\begin{aligned} MML(\theta, D_n, \alpha_i) = &-\sum_{i=1}^{C} \log(\theta_i^{n_i+\alpha_i-\frac{1}{2}}) \\ &+ \frac{C-1}{2}\log n - \frac{C-1}{2}\log 12 \\ &+ \log B_C(\alpha_1, ..., \alpha_C) \end{aligned} \quad (3)$$

where $\theta_i = \frac{n_i+\alpha_i}{n+\alpha_0}$, $\alpha_i (i \neq 0)$ is predefined parameters, $\alpha_0 = \sum_{i=1}^{C} \alpha_i$, $C$ is number of different RHS this LHS has, $n_i$ is the frequency of $i$-th RHS, $n = \sum_{i=1}^{C} n_i$, $B_C()$ is Beta function. Please refer to the Appendix for details.

## III. Experimental study

### A. Royal Tree Problem

Royal Tree Problem [19] is designed to be a difficult problem for GP. In this experiment, we use level-e Royal Tree Problem. This problem has five nonterminals $a, b, c, d, e$ with arity 1,2,3,4,5 respectively and one terminal $x$. The perfect solution for this problem is complete full tree, with nonterminal $e$ as root, only $d$ appear on level 2, only $c$ level 3, and so on and only $x$ on level 6, the deepest level. The fitness is related to the resemblance to this perfect solution and designed to encourage searching this solution in a bottom up manner. The fitness of perfect solution is 122,880. For more detail of Royal Tree Problem, please refer to [19].

In this experiment, the setting for GMPE is population size 60, truncation selection, selection rate $50\%$, $\alpha = 0.5$, maximum depth 6. We conducted 50 runs. The cumulative frequency of successful runs is illustrated in Fig. 2. As we can see, around $80\%$ of runs end before generation 4000 (only $4000 \times 60 = 240,000$ individual evaluations and all runs end roughly at 16,000 generation (only $16,000 \times 60 = 960,000$ individual evaluations).

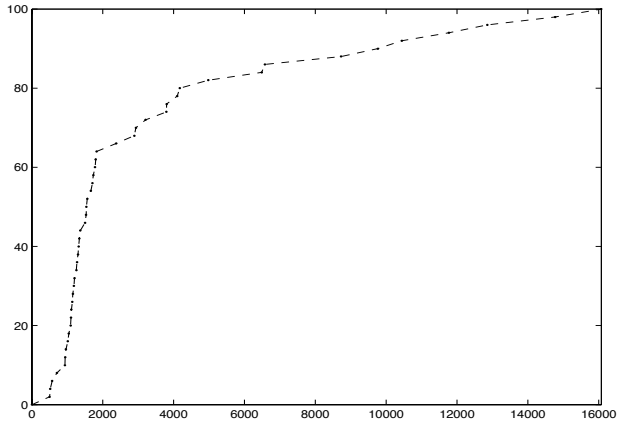


Fig. 2. Cumulative frequency of success measure of GMPE on Royal Tree Problem

Royal Tree Problem is very difficult problem for GP. We present a comparison in Table I to show the relative performance of GMPE. GP statistic of the problem is adopted from [19]. Several GP settings are tried in [19]. We select GP of the best setting, population size 3500, internal crossover 0.875, external crossover 0.075, mutation rate 0.05, maximum depth 17, generation 500, over-selection.

Table I shows that with 1,750,000 evaluations ($population\ size\ 3500 \times generation\ 5000 = 1,750,000$), $50\%$ GP runs succeed. To achieve the same rate of successful runs, GMPE only need 92,460 ($60 \times 1541 = 92,460$) evaluations. We define a simple measure $speedup$, similar to the measure in [11], to give a clearer impression of the relative performance of GMPE to GP. Given rate of successful runs $\delta$,

$$speedup_\delta = \frac{Number\ of\ Evaluations\ of\ GP}{Number\ of\ Evaluations\ of\ GMPE} \quad (4)$$

Therefore, in this experiment, $speedup_{50\%} = \frac{1750000}{92460} \approx 18.9$. Under this measure we say the GMPE is 18.9 times faster in terms of number of evaluations given rate of successful runs $50\%$. We use different maximum depth in GP and GMPE experiments. The nature of the problem make it only possible to discover from small partial solution and then to combine them into bigger partial solution until find the final solution. The failed GP runs did be trapped with local optimum which small partial solution. The difficulty of this problem is to escape from local optimum, not the size of the search space. This suggests the different maximum tree would not have significant impact on the performance of the algorithms. However, we do plan to include fairer comparison in the final version of this paper.

TABLE I
COMPARISON OF GP AND GMPE ON ROYAL TREE PROBLEM. HORIZONTAL AXIS IS THE NUMBER OF GENERATIONS AND THE VERTICAL AXIS IS THE PERCENTAGE OF SUCCESSFUL RUNS.

| Method | Pop | Gen | No. Eval./Run | Succeed | Speedup |
|---|---|---|---|---|---|
| GMPE | 60 | 1541 | 92,460 | 50% | 18.9 |
| GP | 3500 | 500 | 1,750,000 | 50% | |

### B. Max Problem

Max problem [9] has only one terminal $x$ with value 0.5 and two nonterminal $+$ and $\times$. The purpose is to find a tree with maximum fitness under some tree size constraint. In this experiment, we use maximum depth as constraint and set it the maximum depth limit to 7 (root is 1) in this experiment. The maximum fitness is 65536.

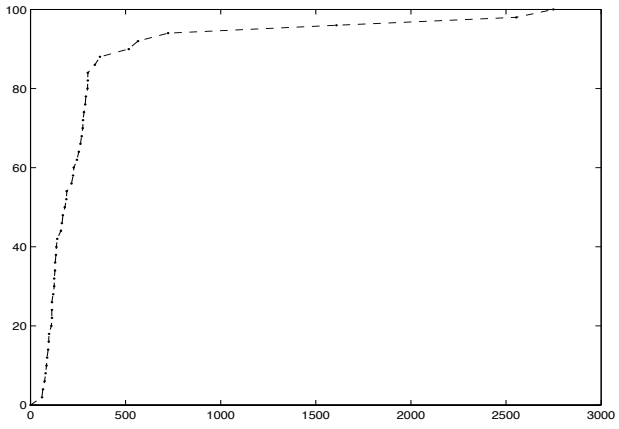


Fig. 3. Cumulative frequency of success measure of GMPE on Max Problem. Horizontal axis is the number of generations and the vertical axis is the percentage of successful runs.

The result of GP runs are from [9]. Briefly, the setting of GP is population size 200, generation size 500, tournament selection, $99.5\%$ crossover, no mutation. In this experiment, the setting for GMPE is population size 60, truncation selection, selection rate $50\%$, $\alpha = 2.0$. We conducted 50 runs. The cumulative frequency of successful runs is illustrated in Fig. 3 and the comparison with GP is shown in Table. II. The $speedup_{60\%}$ is 7.4. Therefore, in this experiment, GMPE is also significantly outperform GP.

TABLE II
COMPARISON OF GP AND MINE ON MAX PROBLEM

| Method | Pop | Gen | No. Eval./Run | Succeed | Speedup |
|---|---|---|---|---|---|
| GMPE | 60 | 224 | 13,440 | 60% | 7.4 |
| GP | 200 | 500 | 1,000,00 | $< 60\%$ | |

## IV. DISCUSSION

Evolving program with tree structure (like GP tree), is far more complex than simply applying EDA, which evolves solution with linear representation, to tree representation. The complexity due to the tree structure is four folded and our method with grammar model can address most of these issues.

1) Tree representation has its internal structure – intrinsic hierarchical structure. The relations/dependencies among parent and child nodes are undoubtedly stronger than among others. When constructing models, this internal structure has to be respected, i.e. the model has to reflect not only the dependencies among the nodes but also the structure constraint. Grammar which is invented to represent internal structure of the language fits exactly this purpose.
2) The semantics of nodes in tree also makes evolving program with GP style tree representation very different from its linear representation counterpart. In evolving linear structure, such as EDA, usually we assume the semantics is attached to the locus. However, in tree representation, The *meaning* of the node has to be interpreted together with its surrounding context. For example, in one of the standard GP benchmark problem – Artificial Ant Problem [8], the node $move$ will have entirely effect depending on the current position of the ant. The model chosen for evolving tree structure has to be able to represent this strong local dependency as well as dependency at larger scale. Context-free Grammar and more expressive grammar can very well model the local and long distance dependencies.
3) In the tree representation, although, in most of circumstances, the meaning of the node is closely related to it surrounding context, it is also common that the building blocks, which are relative independent sub-solutions, are *not*, or at least no strictly, position dependent. For example, a building blocks , which is a subtree in this example, can be written as (+(a, b)). This subtree can occur in position A, B or C in tree (+(+(A,B),C)) and have the same contribution to the overall fitness. A position dependent model, such as PIPE [22] and other prototype tree based work, has to learn building blocks at these positions separately. Grammar model

does not assume any position dependency. The common structures (building block) among individuals, even if they are not at the same position, can be represented and preserved by grammar model.

4) The individual trees have no fixed complexity. Even if we do impose some limitation, such as, maximum depth, maximum number of nodes, the individual complexity significantly varies from individual to individual, from generation to generation. The fixed model, such as prototype tree in [22], which resembles the models in EDA with fixed size in terms of number of variables, cannot reflect this variation. Therefore, the model with fixed complexity might be either at some stage too simple to express all the necessary dependencies or at other stage so complex that requires more time to learn unnecessary dependencies. With proper learning method, the grammar model may very well reflect the superior individuals (training samples) and generalize to certain extent.

We notice that although early work of using model to evolving program with tree representation [22] was published in 1997, the amount of work is entirely incomparable with the amount of work in evolving linear structure, such as EDA. We believe this is due the above mention difficulties. It is clear from the above analysis that our GMPE with grammar model can address most of these difficulties.

## V. Conclusion

In this paper we propose a new method for program evolution. We choose Stochastic Context-free Grammar (SCFG) to model superior tree-form individuals. The new population is generated by sampling this grammar model. Because the grammar model can better present, preserve and promote the common structures in superior individuals, usually called building blocks, than conventional GP, our method significantly outperform conventional genetic programming on two benchmark problems we studies.

The future research issues include more thorough analysis and direct evidence of what has been learned in the grammar, more efficiently grammar learning method, possible extraction of knowledge embedded in the learned grammar, experiments on noisy data.

## Appendix A: Cost of coding probability distribution of SCFG

### A. Dirichlet Prior

A symmetric Dirichlet prior has the form:

$$P(\theta) \propto \prod_{i=1}^{C} \theta_i^{\alpha_i} \tag{5}$$

where $\alpha = 1.0$ corresponds to the uniform prior.

A Dirichlet prior is a conjugate prior because it has the same form as the likelihood. This makes the posterior distribution a simple product of likelihood and prior:

$$P(\theta|D_n) \propto \prod_{i=1}^{C} \theta_i^{n_i+\alpha-1} \tag{6}$$

where $D_n$ are the $n$ observations (e.g. for $C = 2, n = 5$, $D_5 = (0, 1, 1, 0, 0)$ ).

Notice we have used *proportional to* in the two above equations. To make these a proper distribution, we need a normalising term which is a $C$-dimensional beta function:

$$B_C(\alpha_1, ..., \alpha_C) = \frac{\prod_{i=1}^{C} \Gamma(\alpha_i)}{\Gamma(\alpha_0)} \tag{7}$$

where $\alpha_0 = \sum_{i=1}^{C} \alpha_i$.

$$\Gamma(n+1) = n! \tag{8}$$

where $n$ is an integer.

The Dirichlet distribution is then:

$$P(\theta) = \frac{1}{B_C(\alpha_1, ..., \alpha_C)} \prod_{i=1}^{C} \theta_i^{\alpha_i} \tag{9}$$

### B. Costing of coding probability distribution with Dirichlet prior

For number of classes $C$, number of data $n$, the cost of coding this data is:

$$MML(\theta, D_n) = -\log(P(\theta|D_n)P(\theta) + \log\sqrt{\frac{F(C-1)}{12^{C-1}}}) \tag{10}$$

where $F(k)$ is the Fisher Information term.

For Dirichlet distributions, Fisher Information is

$$F(k) = \frac{n^{C-1}}{n + \frac{C}{2}} \tag{11}$$

where $n$ is the number of data items.

Putting it all together gives:

$$\begin{aligned} MML(\theta, D_n, \alpha_i) = &-\sum_{i=1}^{C} \log(\theta_i^{n_i+\alpha_i-\frac{1}{2}}) \\ &+ \frac{C-1}{2}\log n - \frac{C-1}{2}\log 12 \\ &+ \log B_C(\alpha_1, ..., \alpha_C) \end{aligned} \tag{12}$$

where $\theta_i = \frac{n_i+\alpha_i}{n+\alpha_0}$.

### C. Example for Grammar Fragment

Given grammar:

```
A -> B 1
A -> C 3
```

With $\alpha_i = 0.5, C = 2$,

$$
\begin{aligned}
MML = & -\log \frac{1+0.5}{4+1}^{1+0.5-0.5} - \log \frac{3+0.5}{4+1}^{3+0.5-0.5} \\
& + \frac{2-1}{2}\log 4 - \frac{2-1}{2}\log 12 \\
& + \log B_2(0.5, 0.5) \\
& = 1.20397 + 1.07003 + 0.69315 \\
& - 1.24245 + 1.14473 \\
& = 2.86943
\end{aligned}
$$

Throughout the paper, we use base $e$ for logarithm.

## REFERENCES


[1] H. A. Abbass, N. X. Hoai, and R. I. McKay. AntTAG: A new method to compose computer programs using colonies of ants. In *The IEEE Congress on Evolutionary Computation*, pages 1654–1659, 2002.

[2] S. Baluja and S Davies. Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. In *Proc. 1997 International Conference on Machine Learning*, 1997. Also Available as Tech Report: CMUCS -97-107.

[3] Shumeet Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Pittsburgh, PA, 1994.

[4] Jeremy S. de Bonet, Charles L. Isbell, Jr., and Paul Viola. MIMIC: Finding optima by estimating probability densities. In Michael C. Mozer, Michael I. Jordan, and Thomas Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, page 424. The MIT Press, 1997.

[5] R. Etxeberria and P. Larrañaga. Global optimization with bayesian networks. In *Second Symposium on Artifcial Intelligence(CIMAF-99)*, pages 332–339, Cuba, 1999.

[6] G. Harik. Linkage learning via probabilistic modeling in the ECGA. Technical Report IlliGAL Report No. 99010, University of Illinois at Urbana-Champaign, 1999.

[7] G. R. Harik, F. G. Lobo, and D. E. Goldberg. The compact genetic algorithm. *IEEE Transaction on Evolutionary Computation*, 3(4):287–297, November 1999.

[8] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[9] W. B. Langdon and R. Poli. An analysis of the MAX problem in genetic programming. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 222–230, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[10] P. Larrañaga and J. A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academis Publishers, 2001.

[11] Ryszard S. Michalski. Learnable evolution model: Evolutionary processes guided by machine learning. *Machine Learning*, 38:9–40, 2000.

[12] H. Müehlenbein and G. Paaß. From recombination of genes to the estimation of distributions i.binary parameters. In *Lecture Notes in Computer Science 1411: Parallel Problem Solving from Nature, PPSN IV*, pages 178–187. 1996.

[13] H. Müehlenbein and G. Paaß. From recombination of genes to the estimation of distributions i.binary parameters. In *Lecture Notes in Computer Science 1411: Parallel Problem Solving from Nature, PPSN IV*, pages 178–187. 1996.

[14] Heinz Mühlenbein and Thilo Mahnig. The factorized distribution algorithm for additively decompressed functions. In *1999 Congress on Evolutionary Computation*, pages 752–759, Piscataway, NJ, 1999. IEEE Service Center.

[15] Martin Pelikan. *Bayesian optimization algorithm: From single level to hierarchy*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL, 2002. Also IlliGAL Report No. 2002023.

[16] Martin Pelikan, David E. Goldberg, and Erick Cantú-Paz. BOA: The Bayesian optimization algorithm. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, volume I, pages 525–532, Orlando, FL, 13-17 1999. Morgan Kaufmann Publishers, San Fransisco, CA.

[17] Martin Pelikan, David E. Goldberg, and Fernando Lobo. A survey of optimization by building and using probabilistic models. Technical Report IlliGAL Report No. 99018, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Sept 1999.

[18] Martin Pelikan and Heinz Mühlenbein. The bivariate marginal distribution algorithm. In R. Roy, T. Furuhashi, and P. K. Chawdhry, editors, *Advances in Soft Computing - Engineering Design and Manufacturing*, pages 521–535, London, 1999. Springer-Verlag.

[19] William F. Punch, Douglas Zongker, and Erik D. Goodman. The royal tree problem, a benchmark for single and multiple population genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 15, pages 299–316. MIT Press, Cambridge, MA, USA, 1996.

[20] J Rissanen. *Stochastic Complexity in Statistical Inquiry*. World Scientific Press, Singapore, 1989.

[21] Conor Ryan, J. J. Collins, and Michael O Neill. Grammatical evolution: Evolving programs for an arbitrary language. In Wolfgang Banzhaf, Riccardo Poli, Marc Schoenauer, and Terence C. Fogarty, editors, *Proceedings of the First European Workshop on Genetic Programming*, volume 1391, pages 83–95, Paris, 14-15 1998. Springer-Verlag.

[22] R. P. Salustowicz and J. Schmidhuber. Probabilistic incremental program evolution. *Evolutionary Computation*, 5(2):123–141, 1997.

[23] Kumara Sastry and David E. Goldberg. Probabilistic model building and competent genetic programming. Technical Report IlliGAL Report No. 2003013, Illinois Genetic Algorithms Laboratory (IlliGAL), Department of General Engineering,University of Illinois at UrbanaChampaign, 117 Transportation Building,104 S. Mathews Avenue, Urbana, IL 61801, April 2003.

[24] Y. Shan, R. I. McKay, H. A. Abbass, and D. Essam. Program evolution with explicit learning: a new framework for program automatic synthesis. In *Proceedings of 2003 Congress on Evolutionary Computation*, Canberra, Australia, Dec 2003. University College, University of New South Wales, Australia.

[25] Andreas Stolcke. *Bayesian Learning of Probabilistic Language Models*. PhD thesis, University of California, Berkeley, CA, 1994.

[26] C. S. Wallace and D. M. Boulton. An information measure for classification. *The Computer Journal*, 11(2):185–194, 1968.

[27] C. S. Wallace and D. L. Dowe. Minimum message length and kolmogorov complexity. *The Computer Journal*, 42(4):270–283, 1999.

[28] P. A. Whigham. Grammatically-based genetic programming. In Justinian P. Rosca, editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33–41, Tahoe City, California, USA, 9 1995.

[29] P.A. Whigham. Inductive bias and genetic programming. In *Proceedings of First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pages 461–466. UK:IEE, September 1995.

[30] Kohsuke Yanai and Hitoshi Iba. Estimation of distribution programming based on bayesian network. In *Proceedings of Congress on Evolutionary Computation*, pages 1618–1625, Canberra, Australia, Dec 2003.