Grammar model-based program evolution

Conference Paper · July 2004

DOI: 10.1109/CEC.2004.1330895 · Source: IEEE Xplore

6 authors, including:

Robert I. McKay
Australian National University
214 PUBLICATIONS **2,359** CITATIONS
SEE PROFILE

Rohan A. Baxter
Australian Taxation Office
47 PUBLICATIONS **1,423** CITATIONS
SEE PROFILE

Hussein Abbass
UNSW Sydney
391 PUBLICATIONS **5,720** CITATIONS
SEE PROFILE

Daryl Essam
Australian Defence Force Academy
174 PUBLICATIONS **2,634** CITATIONS
SEE PROFILE

Grammar Model-based Program Evolution

Y. Shan, R.I. McKay
School of Info. Tech. and Electr. Eng.
Univ. College, Univ. of New South Wales
ADFA, Canberra ACT 2600
Australia
Email: {shanyin,rim}@cs.adfa.edu.au

R. Baxter
CSIRO ICT Center
GPO 664, Canberra,ACT 2601
Australia
Email: rohan.baxter@csiro.au

H. Abbass, D. Essam, H.X. Nguyen
School of Info. Tech. and Electr. Eng.
Univ. College, Univ. of New South Wales
ADFA, Canberra ACT 2600
Australia
Email: {daryl,abbass,x.nguyen}@cs.adfa.edu.au

***Abstract*— In Evolutionary Computation, genetic operators, such as mutation and crossover, are employed to perturb individuals to generate the 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 is a wide range of such work in Genetic Algorithms; but because of the complexity of the 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 operators with a Probabilistic Context-free Grammar (SCFG). In each generation, an SCFG is learnt, and a new population is generated by sampling this SCFG model. On two benchmark problems we have studied, GMPE significantly outperforms conventional GP, learning faster and more reliably.**

I. Introduction and Related Work

In Evolutionary Computation, genetic operators, such as mutation and crossover, are employed to perturb the individuals and generate the next population. However, these fixed, problem independent genetic operators may destroy the sub-solutions (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. Recently, this kind of research, on EC guided by inductively learnt models, such as the Estimation of Distribution Algorithm (EDA) [1] and the Probabilistic Model-building Genetic Algorithm (PMBGA) [2], has drawn increasing interest.

There are several reasons leading to this increasing interest. Firstly there is the theoretical attraction. The highly complex and dynamic effects of genetic operator are extremely hard to understand and predict. Replacing the genetic operators and population with a well-formed model makes it possible to understand EC. In some simple cases, EC guided by an inductively learnt model is a quite accurate approximation of conventional EC [3], [4]. Secondly, in terms of practical usefulness, empirical studies have demonstrated a superior performance by this kind of method.

Because of the abandonment of genetic operators (either partially or completely), this kind of method behaves very differently from the methods of conventional evolutionary computation. However the representation of individuals is consistent with conventional Genetic Algorithms (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. Consequently, most current research focuses 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, but with no conventional genetic operators. The stochastic grammar model is learnt from the superior individuals in the new population, and the new population is generated by sampling this grammar. A grammar model can represent the common structure of the superior individuals (building blocks) very well. Due to the flexibility of the grammar model, GMPE, the method we propose, far outperforms GP on the two benchmark problems we have studied.

This paper is organized as follows. We briefly review related work with GA style linear representation and then with GP style tree representation in this section, followed by an overview of our GMPE to give the reader a flavour of our method. In Section II, we present our work in detail, including the high level algorithm, the probabilistic model (Stochastic Context-free Grammar), and the model learning method. To validate our idea, two experiments are reported in Section III. We discuss the difficulty of evolving programs with tree representation, compared with evolving solutions with linear representation, such as EDA [1], and how GMPE addresses these difficulties in Section V. The final section is the conclusion.

A. Related Works with GA Style Linear Representation

There is extensive similar work in evolving linear solutions guided by an inductively learnt model; they differ mainly in the models chosen. Among them, we perceive the following two streams.

The first is the Learnable Evolution Model (LEM) [5]. 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, so as to

form inductive hypotheses which explicitly characterize good individuals.

The second is Estimation of Distribution Algorithms (EDA) [6], [7]. 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 [8], [3], [4], through others that take into account pairwise interactions [9], [10], [11], to methods that can accurately model even a very complex problem structure with highly overlapping multivariate building blocks [12], [13], [14], [15].

B. Related Works with GP Style Tree Representation

Because of the complexity of GP style tree representation, the amount of work in evolving programs with tree representation is not comparable with that in approaches dealing with linear representation. We roughly classify GP-based methods into two kinds.

The first kind is probabilistic model based methods. It has a strong connection with EDA, i.e. evolving linear solutions with the guidance of a probabilistic model. It includes the following three projects.

Probabilistic Incremental Program Evolution (PIPE) [16] combines probability vector coding of program instructions, Population-Based Incremental Learning [8], 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 learnt and encoded in the PPT.

Extended Compact Genetic Programmming (ECGP) [17] is a direct application of ECGA [13] in tree representation. Marginal product models (MPMs) are used to model the population of genetic programming. MPMs are formed as a product of marginal distributions 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) [18] tries to model the dependency of adjacent nodes in GP tree. Although there are a few possible dependencies among the adjacent nodes, only the conditional probability of a child node given a parent node is considered in this particular research.

The second kind is grammar-model-based methods. Although a stochastic grammar model is a probabilistic model, it is usually presented in a very specialized literature and it has quite different learning methods. Therefore, for the sake of clarity, we classify it into a different category. Grammar, which is widely used to model the internal hierarchical structure of sentences, is an ideal formalism for modeling GP-style tree structure. The grammar-model-based methods include Whigham's very early work [19], ant-TAG [20] and Program Evolution with Explicit Learning (PEEL) [21].

This second kind of work has some connection with Grammar Guided Genetic Programming (GGGP) [22], [23], [24], i.e. using a 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. In Whigham's work [19], in addition to the normal GGGP search, the grammar is slightly modified during the search. The updated grammar represents the accumulated knowledge found in the process of search.

The other works, ant-TAG and PEEL, are radically different from normal GP and more like EDA style evolution. In both of these works, grammars are updated based on the superior individuals. The new generation is obtained by sampling from the grammars. Conventional GP genetic operators are 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 the fixed grammar means that the search space can only be explored at fixed granularity. PEEL addresses this issue by allowing a change of grammar structure.

C. Overview

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

We replace the common GP genetic operators with a Stochastic Context-free Grammar (SCFG) learning stage. In each generation, an SCFG is learnt, and then a 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 a 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, being both faster and more reliable.

II. Method

A. Algorithm

As in conventional EC, GMPE starts with a randomly generated population. A stochastic grammar model is learnt from the selected individuals of this population. The new generation is generated by sampling the learnt model. Then the next iteration will start again from this new population. The high level algorithm is illustrated as a flow chart in Fig. 1. This structure will be familiar to readers from previous EDA approaches.

In the next subsection, we will focus on the stochastic grammar model, which plays a critical role in GMPE.

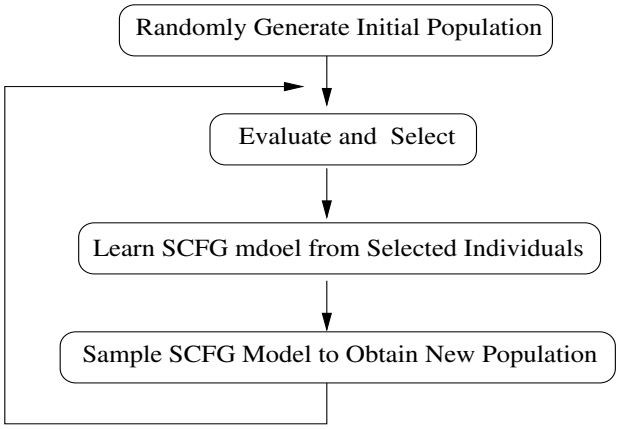

Fig. 1. Flow chart of GMPE

B. Model Learning

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

Previously in GP, grammars, in particular Context-free Grammars (CFG), have been used to constrain the search space [25], [26]. However, since grammar is a formal model for language, both natural and formal language, it can function as more than just the constraint of the search space. If we take the GP individual as a string/sentence, it is not hard to see that the evolved population, corresponding to a 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 elaborate on the definition of SCFG, and how to sample an SCFG to generate the individuals. In II-B.2, the problem of learning a grammar, given a set of superior individuals as training samples, is discussed. The learning method in this work is stochastic hill-climbing search. When searching for grammars, we need a measure to compare grammars. We derive this criterion 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) Σ,
- 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 λ is the righthand side (RHS).

If $X \to \lambda$ is a production of R, then for any strings γ and δ 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 β can be derived from α, denoted $\alpha \overset{*}{\Rightarrow} \beta$, 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 product 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. An LHS may have more than one RHSs. When deriving a string, if we come to this kind of LHS, we need to choose one of its RHSs. SCFG has a probability component attached to each rule (more accurately to each RHS). We choose the RHS based on its probability.

2) Learning method: We use a specific-to-general method to search for a good grammar, motivated by [27]. We start from a very specialized grammar which covers only the training examples (selected superior individuals). The *merge* operator is employed among the rules to generalize the initial grammar. The merge operator take two rules and unifies their LHSs to 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 would be replaced by Y. The reason that the merge operator generalizes the grammar is that before the merge. two rules have different LHSs; after the merge, the two rules share LHSs, so that the merged grammar can cover more strings.

The search strategy in this research is stochastic hill-climbing, i.e. we randomly merge two rules in the grammar, if the score of the 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. Therefore, we need to measure whether a specific merge improves the grammar model. We use minimum length encoding inference, usually referred to as Minimum Message Length (MML) [28], [29] or Minimum Description Length (MDL) [30] to compare grammars. In the remainder of this paper, MML is generally employed to 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. From the perspective of MML, the model (grammar in this case) we are looking for should minimize the cost of coding given the 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 the SCFG in the first part $L(G)$ and then the cost of coding the data (training samples) given the SCFG model in the second part $L(D|G)$.

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

$L(G)$ requires the statement of the inferred SCFG. Although there are some rough estimates of $L(G)$ in the literature, they are not adequate for our purpose as they over-estimate the grammar complexity [27], [31]. In the literature, this over-estimation is compensated by a fudge factor α, which is tuned to fit the data. However direct application of this method in GMPE led to instability in the algorithm, with the size of the generated grammar fluctuating wildly from generation to generation. Theoretical analysis led us to recognize the source of the problem as the inaccurate estimation of grammar complexity. We have derived a more accurate estimate of $L(G)$, eliminating the need for a fudge factor.

To encode an SCFG, we need to encode the names of its 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 omitted from the calculation because the same cost is incurred by all grammars under consideration.
- 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 integers [30]. 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 symbols 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 reflects the encoding of the probabilities of production rules. Some LHSs have more than one RHS. When deriving individuals, we need to know with 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 probability distributions to uniform distribution because this means we would have less uncertainty. This preference can be modeled by a symmetric Dirichlet prior [32]. The last term can be calculated using the following equation.

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

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

III. Experimental study

A. Royal Tree Problem

The Royal Tree Problem [33] was designed to be a difficult problem for GP. In this experiment, we use the level-e Royal Tree Problem. This problem has five nonterminals a, b, c, d and e with arity 1,2,3,4,5 respectively and one terminal x. The perfect solution for this problem is a complete full tree, with nonterminal e as root, only d appearing on level 2, only c on level 3, and so on and only x on level 6, the deepest level. The fitness is proportionate to the resemblance to this perfect solution, and designed to encourage searching for this solution in a bottom up manner. The fitness of the perfect solution is 122,880. For more details of the Royal Tree Problem, please refer to [33].

In this experiment, the settings for GMPE are population size 60, selection rate 50%, maximum depth 6, and $\alpha = \alpha_i = 0.15$ for all i. We use truncation selection, i.e., in each generation, the top 50% of population are selected to build the model and the second 50% are replaced by the individuals generated from the model. To maintain diversity, ten randomly generated individuals are introduced to each generation. We conducted 55 runs.

The cumulative frequency of successful runs is illustrated in Fig. 2. As we can see, around 80% of runs succeed before 2×10^5 individual evaluations and overall around 90% percent of all runs succeed within 6×10^5 individual evaluations.

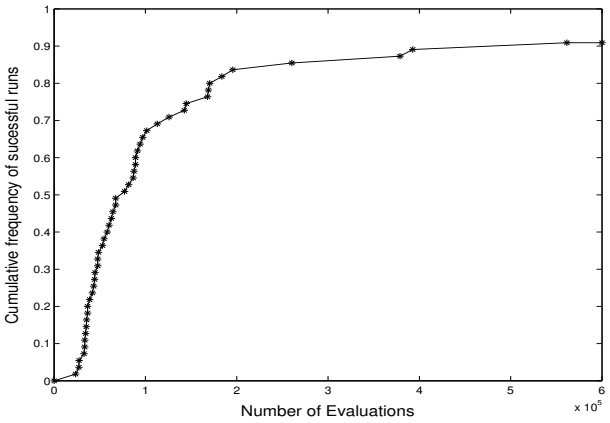

Fig. 2. Cumulative frequency of successful runs of GMPE on Royal Tree Problem. Horizontal axis is the number of individual evaluations and the vertical axis is the percentage of successful runs.

The Royal Tree Problem is a very difficult problem for GP. We present a comparison in Table I to show the relative performance of GMPE. The GP statistics for the problem are drawn from [33]. Several GP settings were tried in [33]. We compare GMPE with the best GP settings, 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 ($population\ size\ 3500 \times generation\ 5000 = 1,750,000$), only 50% of GP runs succeed. To achieve the similar rate of successful runs, GMPE only needs 77,220 ($30 \times 2574 = 77,220$) evaluations. We define a simple measure $speedup$, similar to the measure in [5], to give a clearer impression of the relative performance of GMPE to GP. Given the rate of successful runs δ,

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

In this experiment, $speedup_{50\%} = \frac{1750000}{77220} \approx 22.7$. Under this measure we can say that GMPE is 22.7 times faster, in terms of number of evaluations, for a rate of successful runs = 50%.

We have used a different maximum depth in the GMPE experiments than those used in the GP experiments. The GP experiments used a relatively large maximum depth of 17. Because of the relatively high space complexity of GMPE relative to the tree depth, this maximum depth is infeasible for us. This clearly compromises comparability. However the nature of the problem makes it highly likely that solutions can only be discovered by building from small partial solutions and then combining them into bigger partial solutions until the final solution is found. Certainly, the failed GP runs were trapped in local optima with small partial solutions. That is, the difficulty of this problem for GP is to escape from local optima, not the exploration of a large search space. This suggests that a different maximum tree depth would not have had a significant impact on the performance of the GP algorithms.

TABLE I
COMPARISON OF GP AND GMPE ON ROYAL TREE PROBLEM.

Method	No.Eval/Gen	Gen	No. Eval./Run	Succeed	Speedup
GMPE	30	2574	77,220	50.9%	22.7
GP	3500	500	1,750,000	50%	

B. Max Problem

The Max problem [34] has only one terminal x with value 0.5 and two nonterminals, $+$ and $\times$. The purpose is to find a tree with maximum fitness under some tree size constraint. In this experiment, we use the maximum depth as constraint and set it to 7 (root is 1). The maximum fitness is 65536.

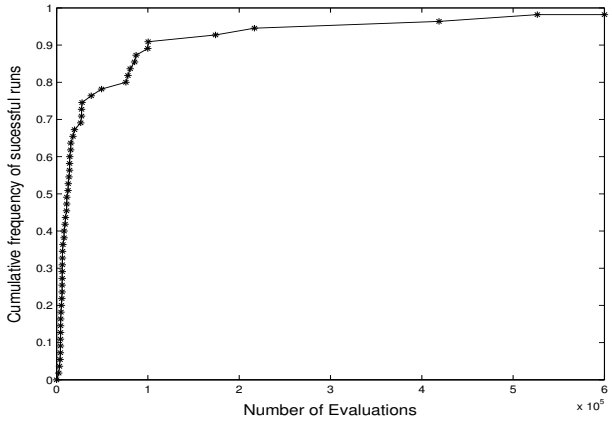

Fig. 3. Cumulative frequency of success runs of GMPE on Max Problem. Horizontal axis is the number of generations and the vertical axis is the percentage of successful runs.

The result of the GP runs are taken from [34]. Briefly, the GP settings are 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%, $\alpha = 0.15$. Ten randomly generated individuals are introduced to each generation. 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. GMPE significantly outperforms GP in this experiment.

TABLE II
COMPARISON OF GP AND MINE ON MAX PROBLEM

Method	No.Eval/Gen	Gen	No. Eval./Run	Succeed	Speedup
GMPE	30	453	13,590	60%	7.4
GP	200	500	100,000	$< 60\%$	

IV. ANALYSIS OF EXPERIMENT RESULTS

From the superior performance, we will form some hypotheses in this section to explain what may be happening during

the GMPE search.

We conjecture that the current version of GMPE can *solve problem constructively in a bottom-up fashion*. In other words, it can efficiently solve the problem

- which has closed subtrees (with no open end) as partial solutions;
- whose final complete solution can be obtained by growing partial solutions.

In GMPE, identical subtrees (with no open end) or building blocks considered in this paper, can be easily identified because merging corresponding rules from identical subtrees only decreases the complexity of the grammar and all the other terms in MML measure are kept intact. Therefore, with simple search methods such as simplified stochastic hill-climbing or other deterministic search methods, the majority or all of identical subtrees can be discovered in GMPE. Since identical subtrees (building blocks) occur multiple times in the population, either in the same individual or among individuals, the rules involved have greater frequencies. Rules with bigger frequencies have greater chances to be preserved in GMPE. Hence, the rules which describe frequently occurred subtrees can be well preserved in GMPE. Then GMPE will do moderate search/variation around these identical subtrees to grow them. This analysis of behavior of GMPE can be clearly validated by the two benchmark problems we have studied.

Royal Tree Problem is a perfect demonstration. Given nonterminal a, b, c, d, e with arity 1,2,3,4,5 and terminal x, the fitness is deliberately designed so that the solution has to be constructed in a bottom-up fashion. The minimum partial solution (a x) has to be firstly identified and then, based on this, a bigger partial solution (b (a x) (a x)) can be constructed and so on. Note due to the carefully designed fitness function, this is the only way to solve this problem. The superior performance of GMPE demonstrates that GMPE does follow a bottom-up fashion in solving Royal Tree problem as we expected.

Max Problem has similar property. One way of solving it is to construct solution in the bottom-up fashion. For example, minimum partial solution is (+ (0.5 0.5)). By combining this minimum partial solution, we can obtain bigger partial solution (+ (+ (0.5 0.5))(+ (0.5 0.5))).

In the current version, we use simplified stochastic hill-climbing as search method. It limits the types of the building blocks that can be discovered. There is no intrinsic difficulty in introducing a more sophisticated searching method. In next Section V, we will discuss GMPE in a more general sense which does not relate GMPE to a specific search method, just assuming an efficient search method has been introduced.

V. Discussion

Evolving programs with tree structure (e.g. GP tree), is far more complex than simply applying EDA, which evolves solutions with linear representation, to tree representation. The complexity due to the tree structure is fourfold; our GMPE approach can address most of these issues.

1) The tree representation has an internal structure – an intrinsic hierarchical structure. The relationships/dependencies between parent and child nodes are undoubtedly stronger than between other nodes. 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. Grammars, which were invented to represent the internal structure of language, exactly fit this purpose.
2) The semantics of nodes in a tree also make evolving a program with GP style tree representation very different from its linear representation counterpart. In evolving a 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 in its surrounding context. For example, in one of the standard GP benchmark problems – the Artificial Ant Problem [35] – the node $move$ will have an entirely different 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 on a larger scale. Context-free (and more expressive) grammars can better model these local and long-distance dependencies.
3) In the tree representation, it is common that the building blocks, which are relatively independent sub-solutions, are *not*, or at least not strictly, position dependent. For example, a max problem building block, 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 [16] and other prototype tree based work, has to learn building blocks at these positions separately. The grammar model does not assume any position dependency. The common structures (building blocks) among individuals, even if they are not at the same position, can be represented and preserved by the grammar model.
4) The individual trees have no fixed complexity. Even if we do impose some limits, such as maximum depth or maximum number of nodes, the individual complexity varies significantly from individual to individual, from generation to generation. Fixed models, such as the prototype tree in [16], which resembles the models in EDA in having fixed size in terms of number of variables, cannot reflect this variation. Because of this, a model with fixed complexity might be at one stage too simple to express all the necessary dependencies, yet at another stage so complex that it requires more time to learn unnecessary dependencies. With an appropriate learning method, the grammar model may very well reflect the superior individuals (training samples) and generalize them to a certain extent.

We note that although the early work using models to evolve programs with tree representation [16] was published in 1997, the subsequent amount of work is entirely incomparable with that in evolving linear structure, such as EDA. We believe this is due the above-mentioned difficulties. It is clear from the analysis that GMPE with a grammar model can address most of these difficulties.

VI. Conclusion

In this paper we propose a new method for program evolution. We use a 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 outperforms conventional genetic programming on the two standard benchmark problems we studied.

Future research issues include

1) more thorough analysis and direct evidence of what has been learnt in the grammar
2) incorporation of more efficient search methods other than simplified hill-climbing,
3) changes to the learning method to directly incorporate learning from previous generations (ie using the previous grammar as a prior), and to incorporate more information from the fitness distribution
4) extraction of knowledge embedded in the learnt grammar
5) experiments on learning from noisy data.

Appendix A: Cost of coding probability distribution of SCFG

A. Dirichlet Prior

A symmetric Dirichlet prior has the form:

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

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:

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

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:

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

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 [36]:

$$MML(\hat{\theta}, D_n, \alpha_i) = -\log \frac{P(\hat{\theta}) P(D_n|\hat{\theta})}{\sqrt{F(\hat{\theta})}} + \frac{C}{2}(1 + \log \frac{1}{12}) \tag{10}$$

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

For Dirichlet distributions, Fisher Information [37] is

$$F(\theta) = \frac{n^{C-1}}{\prod_{i=1}^{C} \theta_i} \tag{11}$$

where n is the number of data items.

Putting it all together gives:

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

where $\hat{\theta}_i = \frac{n_i + \alpha_i}{n + \alpha_0}$.

C. Example for Grammar Fragment

Given grammar:

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

With $\alpha_i = 1.5, C = 2$,

$$\begin{aligned} MML = & -\log \left(\frac{1+1.5}{4+3}\right)^{1.5+1-0.5} - \log \left(\frac{3+1.5}{4+3}\right)^{1.5+3-0.5} \\ & + \log B_2(1.5, 1.5) \\ & + \frac{2-1}{2} \log 4 - \frac{2}{2}(1 + \log \frac{1}{12}) \\ & = 2.05924 + 1.76733 - 0.93471 \\ & + 0.69315 - 1.48491 \\ & = 2.1001 \end{aligned}$$

Throughout the paper, we use base e for logarithm.

REFERENCES

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

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

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

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] Man Leung Wong and Kwong Sak Leung. Evolutionary program induction directed by logic grammars. *Evolutionary Computation*, 5(2):143–180, summer 1997.

[23] 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 July 1995.

[24] Frederic Gruau. On using syntactic constraints with genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 19, pages 377–394. MIT Press, Cambridge, MA, USA, 1996.

[25] 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.

[26] 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.

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

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

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

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

[31] Stanley F. Chen. Bayesian grammar induction for language modeling. In *Meeting of the Association for Computational Linguistics*, pages 228–235, 1995.

[32] W.L. Buntine. *A Theory of Learning Classification Rules*. PhD thesis, School of Computing Science in the University of Technology, Sydney, February 1990.

[33] 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.

[34] 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.

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

[36] C. S. Wallace and P. R. Freeman. Estimation and inference by compact coding. *Journal of the Royal Statistical Society. Series B (Methodological)*, 49(3):240–265, 1987.

[37] Lloyd Allison. Multistate and Multinomial Distributions, 2001. http://www.csse.monash.edu.au/ lloyd/tildeMML/Discrete/Multistate.html.