2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

Grammar-Based Genetic Programming with Bayesian Network

Pak-Kan Wong*, Leung-Yau Lo*, Man-Leung Wong† and Kwong-Sak Leung*
*Department of Computer Science and Engineering
The Chinese University of Hong Kong, Hong Kong
†Department of Computing and Decision Sciences
Lingnan University, Tuen Mun, Hong Kong

***Abstract*—Grammar-Based Genetic Programming (GBGP) improves the search performance of Genetic Programming (GP) by formalizing constraints and domain specific knowledge in grammar. The building blocks (i.e. the functions and the terminals) in a program can be dependent. Random crossover and mutation destroy the dependence with a high probability, hence breeding a poor program from good programs. Understanding on the syntactic and semantic in the grammar plays an important role to boost the efficiency of GP by reducing the number of poor breeding. Therefore, approaches have been proposed by introducing context sensitive ingredients encoded in probabilistic models. In this paper, we propose Grammar-Based Genetic Programming with Bayesian Network (BGBGP) which learns the dependence by attaching a Bayesian network to each derivation rule and demonstrates its effectiveness in two benchmark problems.**

I. Introduction

Many real world problems can be formulated as optimizing a given objective function over a constrained search space. When analytic solutions are not available and exhaustive search is computationally infeasible due to the size of search space, search heuristics become the only way. Genetic Algorithms (GA) [1] and Genetic Programming (GP) [2] are search heuristics inspired by natural selection. GA and GP maintain a population of chromosomes as candidate solutions, and optimizes the objective function through crossover, mutation and reproduction of chromosomes over a number of generations. While the chromosomes in GA are usually fixed-length binary strings, in GP they are usually trees of variable sizes, so the genetic operators in GP operate on sub-trees. One requirement of canonical GP is the closure property (i.e. the domain of the inputs is the same as the domain of the outputs for a sub-tree) on the terminal set and the function set. This property is strong in most problems. For example, when GP is used to build an image classifier from pixels, the pixels and the class labels are in two different domains, hence violating the closure property.

GP can still be done without the closure property if a grammar for the solution is defined and the genetic operators are defined to respect the given grammar. The grammar contains the rules for constructing any syntactically correct solutions. This new heuristic, namely Grammar-Guided Genetic Programming [3][4] or Grammar-Based Genetic Programming (GBGP) [5], marks a new era in the study of GP. It established the connection between GP and grammar. GBGP [6] formalizes constraints during the evolution with grammar. The advantages of GBGP over canonical GP include declarative search space restriction, flexibility in expressing problem structure, homologous operators, and flexible extension. GP is a special case of GBGP in which all the terms in GP are of one type.

In contrast to canonical GA, GP or GBGP, Probabilistic Model-Building Genetic Algorithm (PMBGA) [7] and Probabilistic Model-Building Genetic Programming (PMBGP) [8] focus on estimating a probabilistic distribution of good solutions. In each generation, individuals are generated from the current estimate of the distribution, then a subset of them is selected based on fitness, to update the estimate of the distribution. Therefore, at the end of evolution, not only is a good solution found, but also a good estimate of the distribution of good solution, which gives more insights into the problem nature.

One big issue in GP and GBGP is the high degree of dependence among sub-trees of an individual. In some problems, the optimal sub-tree at one position depends on the choice of sub-tree at another position, and the fitness is high only when the correct sub-trees are chosen at *both* positions. In PMBGP, dependence could be handled by using a probabilistic distribution representation that explicitly models dependence, e.g. Bayesian networks and probabilistic grammar. However, since the trees in GP may have a large number of nodes, modelling the global dependence of these nodes causes difficulty in estimating the distribution from limited samples. Besides, in the grammar model based approaches, the dependence and the non-terminals are often unknown. It may also require a complex grammar to specify the relations. Thus, there is a need of automatic dependence learning.

In this paper, a new GBGP, namely Grammar Based Genetic Programming with Bayesian Network (BGBGP) is proposed to cope with the challenge of dependence. It makes use of Bayesian networks to model the *local* dependence among the non-terminals in each rule and therefore the dependence between sibling sub-trees. It is scalable to large problem and supports partial order in choosing the non-terminals for a rule.

This paper is structured as follows. In the next section, we introduce the existing algorithms of PMBGP and review the basics of Bayesian network. After a formal description of the proposed grammar model in Section III, Section IV outlines the structure of our BGBGP system and discusses the whole system in detail. In Section V, we evaluate our

978-1-4799-1488-3/14/$31.00 ©2014 IEEE

approach on two benchmark problems through experiments. Lastly, we conclude this paper with a discussion and future works in Section VI.

II. Related Works

A. PMBGAs and PMBGPs

Probabilistic Model-Building Genetic Algorithm (PMBGA) or Estimation of Distribution Algorithm (EDA) [9] aims at describing a population of optimal solutions of a problem with a probability distribution. In order to limit the search spaces, it is assumed that a significant change in fitness values is due to huge differences in their structures. Furthermore, as noted in [10], individuals sharing the same components are also assumed to be correlated in their fitnesses. The probabilistic model can be either univariate (UMDA[11], PBIL[12], cGA[13]), bivariate (MIMIC[14], COMIT[15], BMDA[16]) or multivariate (BOA[17], EBNA[18], LDFA[19]).

Inspired by the success in PMBGA, many Probabilistic Model-Building Genetic Programming (PMBGP) approaches have been proposed. There are two classes of models: probabilistic prototype tree (PPT) model-based method and probabilistic grammar model-based method. The former method operates on a fixed-length chromosome, which is essentially a tree structure, and utilizes probabilistic models. Probabilistic Incremental Program Evolution (PIPE)[20], being the first PMBGP, introduced probabilistic prototype tree and adopted a univariate model, which has been subsequently used and extended by Estimation of Distribution Programming (EDP) [21], Extended Compact Genetic Programming (eCGA) [8], BOA programming (BOAP) [22], Program Optimisation with Linkage Estimation (POLE) [23], [24], POLE on binary encoded PPT [25] and POLE-BP [26].

Probabilistic grammar model-based method incorporates probabilistic context-free grammar (PCFG) [27]. Stochastic CFG (SCFG) is a simple PCFGs and adds weights on each derivation rule while the weights on the rules having the same non-terminal are normalized to sum to 1. Stochastic grammar-based GP (SG-GP) [28] combines PBIL [12] with Grammar-Guided Genetic Programming (GGGP) [3] and has been shown to resist bloating. Program evolution with explicit learning (PEEL) [29] extends SG-GP and uses Search Space Description Table (SSDT). It allows the use of depth and the relative location in the tree. Grammar model-based program evolution (GMPE) [30] modifies the grammar to describe the search space by applying stochastic hill-climbing search to find a good merging of the non-terminals of the production rules of SCFG scored by the Minimum Description Length (MDL) [31]. Grammar transformation in an EDA (GT-EDA) [32] is similar to GMPE but it learns from expression trees. Programming with annotated grammar estimation (PAGE) [33] employs PCFG with Latent Annotations (PCFG-LA) to weaken the context free assumption. PAGE-EM uses expectation-maximization (EM) algorithm [34] to estimate the annotations from promising individuals. Variational Bayes (VB) learning [35] was also introduced to PAGE to learn the annotations and infer the number of annotations from the learning data. Unsupervised PAGE (UPAGE) [36] utilizes PCFG-LA mixture model to deal with local dependencies and global contexts. Bayesian automatic programming (BAP) [37] applies a learnt Bayesian network on top of a fixed size chromosome and generates new individuals using the Bayesian network.

Tanev's work was based on Probabilistic Context-Sensitive Grammar (PCSG), which extended PCFG by varying the probability according to the predefined context [38]. In this work, the probabilities were obtained by conditioning on the context which can be extracted from the rules applied before.

Other models have also been proposed. N-gram GP [39] evolves linear GP (that is a machine-language-type program). Ant Tree Adjoining Grammars (AntTAG) [40] evolves programs using an ant colony optimization algorithm and estimates the dependencies in tree structures using the pheromone matrix.

In contrast to the aforementioned PMBGP approaches, we introduce a new PCSG model to maintain the syntactical correctness and overlay each derivation rule with a Bayesian network to keep the number of variables tractable. The Bayesian networks are learnt automatically from the individuals from the past generation which affect the conditional probabilities in the PCSG.

B. Bayesian network

A Bayesian network, also called belief network, is a graphical model to represent the probabilistic independences among variables [41]. It connects probability with graph theory.

Consider a set of random variables $\mathbf{X} = (X_1, X_2, ..., X_n)$ and denote the assignment of states by $\mathbf{x} = (x_1, x_2, ..., x_n)$. We use $\mathbf{X} = \mathbf{x}$ to mean the variable set $\mathbf{X}$ is in configuration $\mathbf{x}$. The joint probability $p(\mathbf{X} = \mathbf{x})$ can be factorized into a set of conditional probability distributions encoded in a Bayesian network, which is a directed acyclic graph. In the network, nodes are the variables and edges represent the conditional dependencies. Each node X_i is associated with a conditional probability table representing the function $p(x_i|\mathbf{pa}(x_i))$, where $\mathbf{pa}(x_i)$ is the configuration of the parents of X_i in the network. The joint distribution for $\mathbf{X}$ can thus be expressed as $p(\mathbf{X} = \mathbf{x}) = \prod_{i=1}^{n} p(x_i|\mathbf{pa}(x_i))$. The Bayesian network provides an explicit representation of the variable dependence and saves memory by exploiting independence and recovering the joint distribution from a set of small conditional distributions.

Very often, the Bayesian network has to be learnt from the training data. Two well-known algorithms for this task are K2 [42] and Max-Min Hill-Climbing (MMHC) [43]. K2 is a greedy algorithm to construct the network. Note that the learnt Bayesian network depends on the input order of the random variables in K2, as each random variable will only form connections with the preceding random variables. MMHC is a constraint-based hill climbing search. The algorithm firstly identifies the possible links in the network using mutual-information, and then orients the edges one by one to obtain the network with local optimal score. Unlike K2, the order of variables need not be given for MMHC.

TABLE I
GRAMMAR FOR DECEPTIVE MAX PROBLEM.

1.0	Start	$\rightarrow$	Exp
2.0	Exp	$\rightarrow$	[*] Exp Exp Exp Exp Exp
2.1		$\rightarrow$	[+] Exp Exp Exp Exp Exp
2.2		$\rightarrow$	[0.95]
2.3		$\rightarrow$	[λ]

TABLE II
EXAMPLE OF FORMALIZED PCSG FOR DECEPTIVE MAX PROBLEM.

N	$\{Start, Exp\}$
T	$\{*, +, 0.95, \lambda\}$
S	$Start$
R	$\{r_0, r_1, r_2, r_3, r_4\}$, where $r_0 = Start \rightarrow Exp$ $r_1 = Exp \rightarrow [*]\ Exp\ Exp\ Exp\ Exp\ Exp$ $r_2 = Exp \rightarrow [+]\ Exp\ Exp\ Exp\ Exp\ Exp$ $r_3 = Exp \rightarrow [0.95]$ $r_4 = Exp \rightarrow [\lambda]$
C	$\emptyset$
D	Initial probability distributions for the rules $p(w, \emptyset \mid r_0) = 1/4$, where $w \in \{r_1, r_2, r_3, r_4\}$ $p(w, \emptyset \mid r_1) = 1/4^5$, where $w \in \{r_1, r_2, r_3, r_4\}^5$ $p(w, \emptyset \mid r_2) = 1/4^5$, where $w \in \{r_1, r_2, r_3, r_4\}^5$

III. GRAMMAR MODEL

A PCSG is employed in our algorithm. Our grammar formulation differs from previous works in the context information used. Formally, a PCSG $\mathcal{G}$ is a 6-tuple consisting of

1) A set of non-terminal symbols N
2) A set of terminal symbols T
3) A start non-terminal symbol $S \in N$
4) A set of derivation rules $R \subset N \times (N \cup T)^*$
5) A set of context elements C
6) A set of derivation probabilities D containing the probabilities $p(w, c|r)$, where $r \in R$ and r contains non-terminal in its right hand side, $L_i^r = \{s \in R : LHS(s) = i^{th}$ element in $RHS(r)\}$, $L^r = L_1^r \times L_2^r \times ... \times L_{|RHS(r)|}^r$, $w \in L^r$ and $c \subseteq C$. In other words, L_i^r is the possible choices of the i^{th} non-terminal and L^r contains all valid combinations to derive the rule r. $p(w, c|r)$ gives the *joint* distribution of the context elements and the expansion choices of the non-terminals for a rule r.

Besides, we define a right-hand side function $RHS : R \rightarrow N^*$ to obtain an ordered set of non-terminals on the right-hand side of a derivation rule. And a left-hand side function $LHS : R \rightarrow N$ gives the left-hand side of a derivation rule. An example of the PCSG for the deceptive max problem [24] is depicted in Table I and the formalized version is given in Table II. We use a pair of square brackets (e.g. [0.95]) to enclose the terminals while the non-terminals (e.g. $Exp, Start$) are not enclosed by them. When the set of context elements C is empty, we may use $p(w|r)$ to represent $p(w, \emptyset|r)$ for notation simplicity. The initial probability distributions for the rules are assumed to be uniformly distributed.

As usual, individuals are represented as trees and constructed according to the choices of the derivation rules at each node. The previous works on PMBGP applied different probabilistic models for the variables (e.g. PIPE: a position-dependent model; POLE: a global dependence model for all nodes in a parse tree). BGBGP has one Bayesian network for each rule that encodes the choice dependency (i.e. $p(w, c|r)$) for it and therefore the sub-trees. In particular, a derivation rule is attached with a Bayesian network B_r encoding the derivation probabilities. For a given derivation rule r, the non-terminal symbols in the left-hand side and the context elements are the nodes in the B_r. Although some non-terminals may appear in several positions in a rule, each non-terminal is regarded as distinct node in B_r. For example, the five Exp non-terminals in Rule 2.0 of the grammar in Table I correspond to five different random variables. Given the grammar, the learning and generation algorithm will then be applied to learn a set of good derivation probabilities of this model (i.e. updating the probability distributions D in Table II) by learning the structures of the Bayesian networks and the conditional probabilities.

IV. PROPOSED ALGORITHM

In this section, we present the details in BGBGP. The whole process of BGBGP involves six steps.

1) Initialize a Bayesian network with independent uniform distribution for each derivation rule.
2) Produce a population of individuals from the grammar according to the probabilities specified in the Bayesian networks.
3) Select the fitter individuals based on the fitness values.
4) Collect the usage counts for all choices of derivation of each production rule from the selected individuals.
5) For each production rule in the grammar, check if the number of samples accumulated exceeds a predefined threshold, if so, its Bayesian network is re-learnt and the samples are cleared.
6) Repeat from step 2 and the selected individuals survive to the next generation.

Step 2 to step 6 are repeated until certain criteria, such as reaching the maximum number of generation, are satisfied.

A. Initialization of Bayesian networks

Initially, all variables are assumed to be independent and thus the nodes in the Bayesian networks are disconnected. For example, given the grammar in Table I and assuming no context element, the initial Bayesian networks (containing nodes only for the non-terminals in the RHS) are shown in Table III.

B. Individual generation

The generation of an individual is the same as deriving a sentence from the grammar except that the derivation procedure of the rules follows the distributions specified by the Bayesian networks. When a non-terminal is to be expanded with a particular rule, the Bayesian network for that rule

TABLE III
BAYESIAN NETWORKS ASSOCIATED GRAMMAR IN TABLE I RIGHT AFTER STEP 1. THE CONDITIONAL PROBABILITY TABLE IS FOR UNIFORM DISTRIBUTION AND IS NOT SHOWN HERE.

Rule	Bayesian network
1.0	Exp_0
2.0	Exp_0 Exp_1 Exp_2 Exp_3 Exp_4
2.1	Exp_0 Exp_1 Exp_2 Exp_3 Exp_4
2.2	No Bayesian network since there is no non-terminal
2.3	No Bayesian network since there is no non-terminal

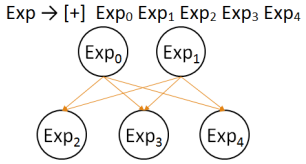

Fig. 1. An example Bayesian Network for rule 2.1 in Table I

is used to instantiate the choices[1] of rule for all the non-terminals, given the context elements, and they can be recursively expanded to get the final tree. As an example shown in Figure 1, we instantiate the rule $r = Exp \rightarrow [+]\ Exp_0\ Exp_1\ Exp_2\ Exp_3\ Exp_4$ according to a topological order in B_r, (i.e. $Exp_0\ Exp_1\ Exp_2\ Exp_3\ Exp_4$ in the example), where the subscript distinguishes the five Exp non-terminals in the RHS. Exp_0 has four possible states representing the four rules (i.e. 2.0, 2.1, 2.2, 2.3) of Exp respectively. To instantiate Exp_0, we choose a state, say state 1, according to the distribution $p(Exp_0)$. This means picking the derivation rule 2.1 of Exp. Similarly, we instantiate Exp_1 with state 3, hence rule 2.3. After the instantiation of Exp_0 and Exp_1, we instantiate Exp_2. As the parents of Exp_2 are now in states 1 and 3 respectively, we look up the conditional probability table for the entries that $Exp_0 = 1$ and $Exp_1 = 3$, draw a value for Exp_2 according to the distribution $p(Exp_2, \emptyset | Exp_0, Exp_1, r)$ and (say) obtain 2. Following the same procedure for Exp_3 and Exp_4, (say) we get 3 and 2 respectively. Therefore, the states are $(1, 3, 2, 3, 2)$ which means the rule is derived to $(2.1, 2.3, 2.2, 2.3, 2.2)$. The chosen rules can then be recursively expanded. Each program can be represented as a parse tree. The chosen rule by the non-terminal will be stored in the parse tree. A valid individual for the grammar in Table I is shown in Figure 2, where $\#n$ in the circles represent the instantiated choice of state for that non-terminal is n.

C. Bayesian networks learning

Since each node in the parse tree records the rule choice, it is possible to acquire the set of applied rules in constructing an individual. In the example in Figure 2, we know that the $Start$ applies rule 1.0, and the children Exp is in turn instantiated to state 1 and the non-terminals of right-hand side of the rule $r_{2,1}$ are being instantiated to states $(1, 3, 2, 3, 2)$. We collect the derivation events, and construct a table for each derivation rule. With this table of cases, Bayesian network learning algorithms can be used to model the relations of variables in the network. In this work, the K2 algorithm is chosen to learn the structure of the Bayesian network for the variables [42], where the input order of the nodes is: the context elements come first, then the nodes for non-terminals with the same order as in the grammar rule. Since we need enough samples to build a better Bayesian network, a parameter $accumulation\text{-}size$ is defined and is usually small relative to the population size. Only when the number of cases is equal to or greater than the $accumulation\text{-}size$, the Bayesian network is re-learnt.

D. Context elements during derivation

In our proposed grammar model in Section III, we can easily add extra contextual information to the conditional probabilities. In the Bayesian network, a new variable node corresponding to each context element will be added (preceding the nodes for the non-terminals). In this paper, we have introduced three different context elements to boost the performance of BGBGP.

- **Depth**: When a rule is derived, it may be beneficial if the non-terminals are chosen according to the depth of the current level. For example, in Figure 3, for the bottom layer of non-terminals, the state of $Depth$ context is 2, so the choices of the five Exp non-terminals will be conditioned on $Depth = 2$.
- **Caller**: In the derivation of a parse tree, a set of rules are chosen according to the grammar. Not only the derivation can be affected by other non-terminals within the same rule, but also affected by whom (the parent rule) calling the rule.
  It is observed that a non-terminal symbol appears in a subset of rules. Consider a non-terminal symbol s, the corresponding subset of rules $Z \subseteq R$ are rules containing s in their RHS. We index these rules (starting from 0 to $|Z|$-1) by their order of occurrence in the grammar. Then, we subsequently annotate the non-terminals having symbol s on the right-hand side of each rule in Z using the index, namely the $caller$ value. For example, considering the Exp symbol in Figure 4, Rule 1.0, Rule 2.0 and Rule 2.1 are indexed by 0,1 and 2 respectively with respect to the symbol. Next, the Exp non-terminals on the right-hand side of Rule 1.0 (having index 0) are annotated by 0 as shown using superscript in Exp^0 and similarly for other rules. We repeat this for all existing non-terminal symbols on the right-hand side in the grammar. Subsequently, each non-terminal is associated with a $caller$ value.
- **Call position**: We annotate globally instances of the same non-terminal in the grammar. In other words, we regard every non-terminal as distinct instance in a global manner. For example, in Figure 5, the Exp^3 in the third rule is

[1]Each rule has a minimum derivable depth. The allowed rules are those where the sum of the current depth and the minimum derivable depth does not exceed the maximum tree depth.

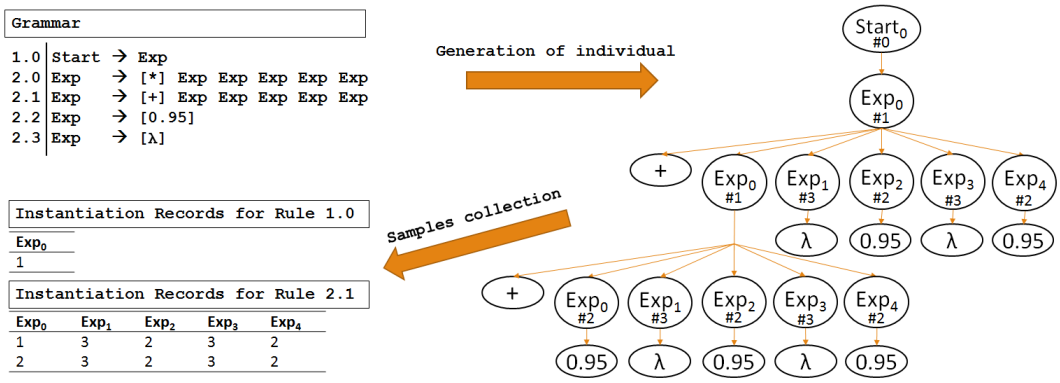

Fig. 2. A valid parse tree on the deceptive max grammar in Table I

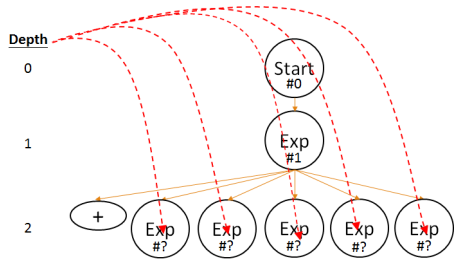

Fig. 3. Tree depth context.

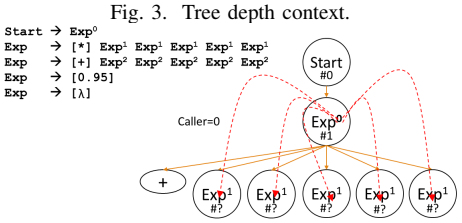

Fig. 4. Caller context.

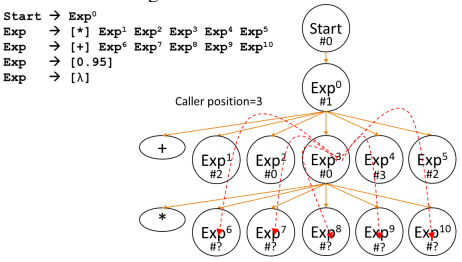

Fig. 5. Call position context.

the fourth Exp in the right-hand side in the grammar so this instance of non-terminal Exp has the call position 3 when it is expanded.

These elements are represented as variable nodes in the network. Since contextual information is provided during the derivation procedure, they can solely be the parent of other nodes.

V. Comparative Experiments

In order to demonstrate the capability of our algorithm, BGBGP was compared with canonical GBGP and three state-of-the-art GP methods: POLE[23], [24], PAGE-EM [33], and PAGE-VB[33]. These three methods were shown to be superior to univariate model (PIPE), adjacent model (EDP model) and simple GP [33]. These approaches were applied to two benchmark problems: the deceptive maximum (DMax) and the royal tree (RT) problems. The parameters and their values used in the system configuration and problems are shown in Table IV. Due to time constraints, if a method cannot obtain an optimal solution in 200,000 fitness evaluations, we assume it fails for the run. We have tested all eight combinations of the context variables: depth *depth*, caller *caller* and call position *callpos*. We label BGBGP with *depth* if depth context variable is used and similarly for other variables. Besides, '-' is used if more than one variable is used. For example, *caller-callpos* indicates that *caller* and *callpos* variables are used. If none of the context variables is used, we call it a *plain* BGPGP.

A. Deceptive Max Problem

The objective in the Max problem [44] is to find a function that returns the largest value within the maximum tree depth. Since this problem is very easy to solve, the authors in [24] formulated a deceptive version of it, called the Deceptive Max (DMax) problem. The interaction of the values needs to be considered. The function set is $F = \{+_m, \times_m\}$ and the terminal set is $T = \{\lambda, 0.95\}$ where $\lambda^r = 1$, $\lambda \in \mathcal{C}$ and $r \in \mathcal{N}$ (so λ is a primitive r^{th} root of unity), $+_m(a_0, a_1, ..., a_{m-1}) = \sum_{i=0}^{m-1} a_i$ and $\times_m(a_0, a_1, ..., a_{m-1}) = \prod_{i=0}^{m-1} a_i$. This problem is difficult since the fitness value only considers the real part of a complex value while the imaginary part is completely ignored but multiplication on imaginary parts may change the magnitude of the real part.

The DMax problem uses $m = 5, r = 3$. The optimal solution is $(5\lambda)^3(0.95 \times 5)^2 = 2820.3125$ for depth 3. For depth 4, the optimal solution is $(5\lambda)^{24}(0.95 \times 5) \approx 2.83 \times 10^{17}$. The parameters used in this experiment are shown in Table IV. The results are depicted in Table V. Following the procedures in previous works [25][33], we report the average number of fitness evaluations needed to obtain the optimal solution with a probability of at least 90% in 50 independent runs under different configurations. Some results of POLE, PAGE-EM, and PAGE-VB are adopted from the papers [25][33] and the

TABLE IV
PARAMETER VALUES FOR TWO BENCHMARK PROBLEMS.

Problem Specific Configuration	
Parameter	**Value**
Depth	DMax: 3, 4 RT: 5 (Level D), 6 (Level E)
GBGP Configuration	
Parameter	**Value**
Population size	1000
Generation	1000
Crossover rate	0.9 (DMax,RT)
Mutation rate	0.09 (DMax,RT)
POLE Configuration	
Parameter	**Value**
Population size	4000 (DMax), 1000 (RT)
Parent range	2
Selection rate	0.1
Elite rate	0.005
PAGE-EM Configuration	
Parameter	**Value**
Population size	1000 (RT level D, DMax) 2500 (RT level E)
Annotation size	2, 4, 8, 16 (RT level D) 16 (RT level E) 8 (DMax)
Selection rate	0.1 (RT level D, DMax) 0.05 (RT level E)
Elite rate	0.1 (RT level D, DMax) 0.05 (RT level E)
PAGE-VB Configuration	
Parameter	**Value**
Population size	1000 (RT level D, DMax) 4000 (RT level E)
Annotation to explore	{1,2,4,8,16}
Annotation exploration range	2
Selection rate	0.1 (RT level D, DMax) 0.05 (RT level E)
Elite rate	0.1 (RT level D, DMax) 0.05 (RT level E)
BGBGP Configuration	
Parameter	**Value**
Population size	500,1000
Accumulation size	10, 25
Context variables tested	Depth, Call position, Caller
Selection rate	0.1

numbers in square brackets are the corresponding references. For the BGBGP tests, only the results with $accumulation\text{-}size$ of 25 and population size 1000 are presented because the difference caused by varying either the $accumulation\text{-}size$ or the population size is insignificant. It can be observed that BGBGP outperforms POLE in the DMax problem at depth 4 (POLE takes up to 7 times more evaluations) when some context elements have been used and has a similar value of the average number of fitness evaluations as in PAGE-EM. In order to construct an optimal solution, the two building blocks (5λ and 5×0.95) must be constructed, which can be done by correctly deriving the Exp non-terminals in Rule 2.1. Table VI gives the Bayesian network for Rule 2.1 ($Exp \rightarrow [+]$ Exp_0 Exp_1 Exp_2 Exp_3 Exp_4) in the last stage of BGBGP with *caller* context. When this rule is invoked by Rule 2.0, this gives $caller = 1$. Once Exp_0 is in state 3, Exp_1, Exp_2, Exp_3 and Exp_4 have a high chance of being in state 3, and thus producing 5λ. Similarly, 5×0.95 can be constructed with high probability (See the bolded numbers in Table VI). Thus, BGBGP successfully estimates the distributions for constructing the substructures.

TABLE V
RESULTS FOR THE DMAX PROBLEM. NUMBERS IN BRACKETS MEAN IT FAILS TO OBTAIN THE OPTIMAL SOLUTION WITH A PROBABILITY OF AT LEAST 90% IN 50 INDEPENDENT RUNS. 'X' MEANS NONE OF THE RUNS SUCCEEDS., '-' MEANS RESULTS ARE UNAVAILABLE.

	Depth 3		Depth 4	
	μ	σ	μ	σ
BGBGP				
-plain	(4,820)	(715)	(93,982)	(28,317)
-depth	4,431	434	15,448	5,456
-callpos	4,372	591	16,213	4,185
-caller	4,226	493	14,936	4,683
-depth-callpos	4,378	496	15,158	2,835
-callpos-caller	4,082	392	14,432	4,876
-depth-caller	4,306	519	**14,108**	2,754
-depth-callpos-caller	4,397	511	16,110	7,882
GBGP	(25,878)	(545)	x	x
POLE	**1,499**[25]	83[25]	119,572[33]	4,295[33]
PAGE-EM	5,040	450	16,043[33]	1,189[33]
PAGE-VB	5,880	659	18,293[33]	1,688[33]

TABLE VI
THE LEARNT BAYESIAN NETWORK FOR RULE 2.1 TO GENERATE A GLOBAL OPTIMAL INDIVIDUAL IN BGBGP WITH *caller* VARIABLE.

$caller$	0	1	2	
$p(caller)$	0.996	0.00199	0.00199	
Exp_0	0	1	2	3
$p(Exp_0 \mid caller = 0)$	0.000200	0.000200	0.174	0.826
$p(Exp_0 \mid caller = 1)$	0.25	0.25	0.25	0.25
$p(Exp_0 \mid caller = 2)$	0.25	0.25	0.25	0.25
Exp_1	0	1	2	3
$p(Exp_1 \mid Exp_0 = 0)$	0.25	0.25	0.25	0.25
$p(Exp_1 \mid Exp_0 = 1)$	0.25	0.25	0.25	0.25
$p(Exp_1 \mid Exp_0 = 2)$	0.00114	0.00114	**0.903**	0.0950
$p(Exp_1 \mid Exp_0 = 3)$	0.000242	0.000242	0.0210	**0.978**
Exp_2	0	1	2	3
$p(Exp_2 \mid Exp_1 = 0)$	0.25	0.25	0.25	0.25
$p(Exp_2 \mid Exp_1 = 1)$	0.25	0.25	0.25	0.25
$p(Exp_2 \mid Exp_1 = 2)$	0.00114	0.00114	**0.860**	0.138
$p(Exp_2 \mid Exp_1 = 3)$	0.000242	0.000242	0.0758	**0.924**
Exp_3	0	1	2	3
$p(Exp_3 \mid Exp_0 = 0)$	0.25	0.25	0.25	0.25
$p(Exp_3 \mid Exp_0 = 1)$	0.25	0.25	0.25	0.25
$p(Exp_3 \mid Exp_0 = 2)$	0.00114	0.00114	**0.775**	0.223
$p(Exp_3 \mid Exp_0 = 3)$	0.000242	0.000242	0.0597	**0.940**
Exp_4	0	1	2	3
$p(Exp_4 \mid Exp_0 = 0)$	0.25	0.25	0.25	0.25
$p(Exp_4 \mid Exp_0 = 1)$	0.25	0.25	0.25	0.25
$p(Exp_4 \mid Exp_0 = 2)$	0.00114	0.00114	**0.811**	0.186
$p(Exp_4 \mid Exp_0 = 3)$	0.000242	0.000242	0.0607	**0.939**

B. Royal Tree Problem

In the royal tree problem [45], the set of functions is $\{A, B, C, ...\}$ and the terminal set is $\{x\}$. Each function has an incremental arity (A is unary, B is binary, C is ternary and so on). The optimal solution is the perfect tree which has as children perfect trees of one level smaller. For example,

the perfect tree at level C has three perfect trees of level B. At level A, the perfect tree has A as root, and has only one terminal x as child. The score of the tree is given by the score of the root node and $Score(x) = 1$. The score of a node X_i is computed by the following equation (1)

$$Score(X_i) = wb_i \sum_j wa_{ij} \times Score(X_{ij}) \quad (1)$$

where X_{ij} is the j^{th} sub-tree of X_i counting from the left. Children weight wa_{ij} is defined as follows:

- Full bonus = 2, if a sub-tree rooted at X_{ij} has a correct root with respect to the root node at X_i and is a perfect tree
- Partial bonus = 1, if a sub-tree rooted at X_{ij} has a correct root with respect to the root node at X_i but is not a perfect tree
- Penalty = 1/3, if a sub-tree rooted at X_{ij} is not a correct root with respect to the root node at X_i

Root weight wb_i is defined as follows:

- Complete bonus = 2, if the tree rooted at X_i is a perfect tree
- Otherwise = 1

Also, $Score(x) = 1$.

The results of the royal tree problem are presented in Table VII which shows the average number of fitness evaluations for obtaining the optimal solution in 50 runs under different configurations. Some results of PAGE-EM and PAGE-VB are adopted from the papers [33][36] and the numbers in square brackets are the corresponding references as shown in the table. For the BGBGP tests, only the results with $accumulation\text{-}size$ of 25 and population size 1000 are presented because of the little difference in results in changing these parameters. BGBGP with the context-element(s) *detph, depth-callpos, depth-caller,* and *depth-callpos-caller* settings significantly outperforms PAGE-EM and PAGE-VB. BGBGP shows its advantage when depth context is included in which it is six times faster because the choice of the rules heavily depends on the current level in order to generate a perfect tree. Besides, as the royal tree becomes more complex, BGBGP can be more scalable than PAGE-EM and PAGE-VB. We terminate POLE for the level E experiment because it takes two hours to complete one generation in our machines, which is significantly longer (usually within one minute for other approaches). The bottleneck of POLE is in the reconstruction of Bayesian network since the number of nodes of each Bayesian network grows rapidly when the tree depth and arity of the new function node both increase using the Expanded Parse Tree. From these results, BGBGP has the potential to perform better than other methods when the problem has strong dependence of depth.

TABLE VII
RESULTS FOR ROYAL TREE.

	Level D		**Level E**	
	μ	σ	μ	σ
BGBGP				
-plain	6,047	631	51,832	12,887
-depth	3,411	267	10,163	1,000
-callpos	5,351	430	38,759	10,181
-caller	4,851	632	23,824	4,413
-depth-callpos	3,401	307	**9,957**	839
-callpos-caller	4,812	693	23,942	4,432
-depth-caller	3,303	311	9,977	983
-depth-callpos-caller	**3,342**	310	**9,957**	861
GBGP	(33716)	(0)	x	x
POLE	12,720	1,126	-	-
PAGE-EM	6,237[36]	18[36]	61,809[33]	15,341[33]
PAGE-VB	11,240	7,631	263,720[33]	63,528[33]

VI. DISCUSSION AND CONCLUSION

From the experiment results, dependence learning improves the performance of BGBGP for problems requiring strong sub-tree dependence. Besides, it has several nice properties.

A. Partial order in instantiation

Tanev's approach [38] which tries to accumulate context information using the derived terms, where sampling enough data can be a problem. Besides, it relies on the derivation to be in a specific order (say in a depth-first search manner and from left-to-right). In contrast, BGBGP does not require a predefined instantiation order. Instead, the order is determined by the dependence of instantiating the non-terminals. Conceptually, if the non-terminals do not depend on other non-terminals, it can be any valid value so they can be instantiated first and alter the probabilities of instantiation of the remaining non-terminals that depend on them.

B. Automatically discard independent events

Instead of blindly associating context variables with the non-terminals, the dependence between them is learnt from data, and is therefore adaptive to the problem nature. If the non-terminals do not depend on the context elements, the values of the context information do not affect the assignment of them. In other words, irrelevant information can be automatically discarded when appropriate through the Bayesian network learning.

C. More scalable

One limitation of using a Bayesian network approach is the scalability of Bayesian network itself. In practice, large scale Bayesian network learning is still an active research problem. Our approach handles this issue by localizing a Bayesian network to a single rule, usually having a smaller number of nodes and thus much simpler. Therefore, our system should be more scalable than eCGP, PEEL, and POLE.

D. Future work

A novel GBGP system named BGBGP has been proposed to handle the dependence issue. It employs multiple Bayesian networks to accelerate the discovery of optimal solution using context sensitive information. It performs well in the Deceptive Max and Royal Tree problems. In the future, we will also study how control parameters influence the search power. The future extensions of BGBGP can learn more sophisticated

global dependence and be applied to real world problems. For example, an optimal solution of the Bi-polar Royal Tree problem [24] requires all the leaf nodes to be the same (i.e. either x or y) but our current framework cannot collect the context information across different branches located in other levels. We would like to identify a set of commonly occurring context elements to tackle the complex dependence in real world problems.

VII. Acknowledgement

This research is supported by General Research Fund LU310111 from the Research Grant Council of the Hong Kong Special Administrative Region.

References

[1] J. H. Holland, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[2] J. R. Koza, *Genetic Programming: vol. 1, On the programming of computers by means of natural selection*. MIT press, 1992, vol. 1.

[3] P. A. Whigham, "Grammatically-based genetic programming," in *Proc. the workshop on genetic programming: from theory to real-world applications*, vol. 16, no. 3, 1995, pp. 33–41.

[4] W. Bohm and A. Geyer-Schulz, "Exact uniform initialization for genetic programming," *Foundations of Genetic Algorithms IV*, pp. 379–407, 1997.

[5] R. I. Mckay, N. X. Hoai, P. A. Whigham, Y. Shan, and M. ONeill, "Grammar-based genetic programming: a survey," *Genetic Programming and Evolvable Machines*, vol. 11, no. 3-4, pp. 365–396, 2010.

[6] M. L. Wong and K. S. Leung, "Combining genetic programming and inductive logic programming using logic grammars," in *Evolutionary Computation, 1995., IEEE International Conference on*, vol. 2. IEEE, 1995, pp. 733–736.

[7] M. Pelikan and D. E. Goldberg, *Bayesian optimization algorithm: From single level to hierarchy*. University of Illinois at Urbana-Champaign Urbana, IL, 2002.

[8] K. Sastry and D. E. Goldberg, "Probabilistic model building and competent genetic programming," in *Genetic Programming Theory and Practice*. Springer, 2003, pp. 205–220.

[9] P. Larrañaga and J. A. Lozano, *Estimation of distribution algorithms: A new tool for evolutionary computation*. Springer, 2002, vol. 2.

[10] K. Kim, Y. Shan, X. H. Nguyen, and R. McKay, "Probabilistic model building in genetic programming: a critical review," *Genetic Programming and Evolvable Machines*, pp. 1–53, 2013.

[11] H. Mühlenbein, J. Bendisch, and H.-M. Voigt, "From recombination of genes to the estimation of distributions ii. continuous parameters," in *Parallel Problem Solving from NaturePPSN IV*. Springer, 1996, pp. 188–197.

[12] S. Baluja, "Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning," DTIC Document, Tech. Rep., 1994.

[13] G. R. Harik, F. G. Lobo, and D. E. Goldberg, "The compact genetic algorithm," *IEEE Trans. on Evolutionary Computation*, vol. 3, no. 4, pp. 287–297, 1999.

[14] J. S. De Bonet, C. L. Isbell, and P. Viola, "Mimic: Finding optima by estimating probability densities," *Advances in neural information processing systems*, pp. 424–430, 1997.

[15] S. Baluja and S. Davies, "Combining multiple optimization runs with optimal dependency trees," 1997.

[16] M. Pelikan and H. Mühlenbein, "The bivariate marginal distribution algorithm," in *Advances in Soft Computing*. Springer, 1999, pp. 521–535.

[17] M. Pelikan, "Bayesian optimization algorithm," in *Hierarchical Bayesian Optimization Algorithm*. Springer, 2005, pp. 31–48.

[18] R. Etxeberria and P. Larranaga, "Global optimization using bayesian networks," in *Second Symposium on Artificial Intelligence (CIMAF-99)*, 1999, pp. 332–339.

[19] H. Mühlenbein and T. Mahnig, "FDA-A scalable evolutionary algorithm for the optimization of additively decomposed functions," *Evolutionary computation*, vol. 7, no. 4, pp. 353–376, 1999.

[20] R. Salustowicz and J. Schmidhuber, "Probabilistic incremental program evolution," *Evolutionary Computation*, vol. 5, no. 2, pp. 123–141, 1997.

[21] K. Yanai and H. Iba, "Estimation of distribution programming based on bayesian network," in *Proc. the IEEE Congress on Evolutionary Computation*, vol. 3. IEEE, 2003, pp. 1618–1625.

[22] M. Looks, B. Goertzel, and C. Pennachin, "Learning computer programs with the bayesian optimization algorithm," in *Proc. the IEEE Congress on Evolutionary Computation*. ACM, 2005, pp. 747–748.

[23] Y. Hasegawa and H. Iba, "Estimation of bayesian network for program generation," in *Proc. 3rd Asian-Pacific Workshop on Genetic Programming*. Citeseer, 2006, p. 35.

[24] ——, "A bayesian network approach to program generation," *IEEE Trans. on Evolutionary Computation*, vol. 12, no. 6, pp. 750–764, 2008.

[25] T. Yanase, Y. Hasegawa, and H. Iba, "Binary encoding for prototype tree of probabilistic model building GP," in *Proc. the GECCO 2009*, 2009, pp. 1147–1154.

[26] H. Sato, Y. Hasegawa, D. Bollegala, and H. Iba, "Probabilistic model building GP with belief propagation," in *Proc. the IEEE Congress on Evolutionary Computation*. IEEE, 2012, pp. 1–8.

[27] T. L. Booth and R. A. Thompson, "Applying probability measures to abstract languages," *IEEE Trans. on Computers*, vol. 100, no. 5, pp. 442–450, 1973.

[28] A. Ratle and M. Sebag, "Avoiding the bloat with stochastic grammar-based genetic programming," in *Artificial Evolution*. Springer, 2002, pp. 255–266.

[29] R. Shan, R. McKay, H. Abbass, and D. Essam, "Program evolution with explicit learning," in *Proc. the IEEE Congress on Evolutionary Computation*, vol. 3. IEEE, 2003, pp. 1639–1646.

[30] Y. Shan, R. I. McKay, R. Baxter, H. Abbass, D. Essam, and H. Nguyen, "Grammar model-based program evolution," in *Proc. IEEE Congress on Evolutionary Computation*, vol. 1. IEEE, 2004, pp. 478–485.

[31] R. Solomonoff, "Complexity-based induction systems: comparisons and convergence theorems," *IEEE Trans. on Information Theory*, vol. 24, no. 4, pp. 422–432, 1978.

[32] P. A. Bosman and E. D. de Jong, "Grammar transformations in an EDA for genetic programming," *UU-CS*, no. 2004-047, 2004.

[33] Y. Hasegawa and H. Iba, "Latent variable model for estimation of distribution algorithm based on a probabilistic context-free grammar," *IEEE Trans. on Evolutionary Computation*, vol. 13, no. 4, pp. 858–878, 2009.

[34] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 1–38, 1977.

[35] H. Attias, "Inferring parameters and structure of latent variable models by variational bayes," in *Proc. the Fifteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1999, pp. 21–30.

[36] Y. Hasegawa and S. Ventura, "Programming with annotated grammar estimation," *Genetic Programming-New Approaches and Successful*, pp. 49–74, 2012.

[37] E. N. Regolin and A. T. R. Pozo, "Bayesian automatic programming," in *Genetic Programming*. Springer, 2005, pp. 38–49.

[38] I. Tanev, "Incorporating learning probabilistic context-sensitive grammar in genetic programming for efficient evolution and adaptation of snakebot," in *Genetic Programming*. Springer, 2005, pp. 155–166.

[39] R. Poli and N. F. McPhee, "A linear estimation-of-distribution GP system," in *Genetic Programming*. Springer, 2008, pp. 206–217.

[40] H. A. Abbass, X. Hoai, and R. I. Mckay, "AntTAG: A new method to compose computer programs using colonies of ants," in *Proc. the IEEE Congress on Evolutionary Computation*, vol. 2. IEEE, 2002, pp. 1654–1659.

[41] D. Heckerman, *A tutorial on learning with Bayesian networks*. Springer, 2008.

[42] G. F. Cooper and E. Herskovits, "A bayesian method for the induction of probabilistic networks from data," *Machine learning*, vol. 9, no. 4, pp. 309–347, 1992.

[43] I. Tsamardinos, L. E. Brown, and C. F. Aliferis, "The max-min hill-climbing bayesian network structure learning algorithm," *Machine learning*, vol. 65, no. 1, pp. 31–78, 2006.

[44] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," *Statistics and Computing*, vol. 4, no. 2, pp. 87–112, 1994.

[45] W. F. Punch, "How effective are multiple populations in genetic programming," *Genetic Programming*, vol. 98, pp. 308–313, 1998.