SCIENCE CHINA
Information Sciences

• RESEARCH PAPERS •

December 2011 Vol. 54 No. 12: 2544–2553
doi: 10.1007/s11432-011-4411-8

Modeling grammatical evolution by automaton

HE Pei[1,3]*, Colin G. JOHNSON[2] & WANG HouFeng[4]

[1]*State Key Laboratory of Software Engineering, Wuhan University, Wuhan* 430072*, China;*
[2]*School of Computing, University of Kent, Canterbury,* CT2 7NF*, England;*
[3]*School of Computer and Communication Engineering, Changsha University of Science and Technology, Changsha* 410114*, China;*
[4]*Institute of Computational Linguistics, Peking University, Beijing* 100080*, China*

Received August 4, 2010; accepted January 14, 2011

Abstract Twelve years have passed since the advent of grammatical evolution (GE) in 1998, but such issues as vast search space, genotypic readability, and the inherent relationship among grammatical concepts, production rules and derivations have remained untouched in almost all existing GE researches. Model-based approach is an attractive method to achieve different objectives of software engineering. In this paper, we make the first attempt to model syntactically usable information of GE using an automaton, coming up with a novel solution called model-based grammatical evolution (MGE) to these problems. In MGE, the search space is reduced dramatically through the use of concepts from building blocks, but the functionality and expressiveness are still the same as that of classical GE. Besides, complex evolutionary process can visually be analyzed in the context of transition diagrams.

Keywords genetic programming, grammatical evolution, finite state automaton, model

Citation He P, Johnson C G, Wang H F. Modeling grammatical evolution by automaton. Sci China Inf Sci, 2011, 54: 2544–2553, doi: 10.1007/s11432-011-4411-8

1 Introduction

Grammatical evolution (GE) was developed in the work of O'Neill et al. [1–3], by combining the search technique of genetic algorithms [4] with a context free grammar [5, 6] to represent the program search space. The major difference between GE and canonical genetic programming (GP) [7–9] is that in GE a context free grammar and a special genotype-to-phenotype mapping have been employed to interpret a string of codons (integers in [0, 255], usually represented as 8 bits), called the genotype, as certain sentential forms. Once sentences are successfully evolved in this system, so are the functional phenotypic programs. This approach has now been applied and validated practically in many areas [1, 3, 10–13], including financial prediction, pattern recognition, machine learning, robot control, and caching algorithms, etc.

In principle, GE is very simple, describing individuals in terms of genotypes and phenotypes (syntactically correct programs in the language of interest), and generating a computer program via the following canonical method, called classical grammatical evolution (CGE or GE for short in the following discussion).

*Corresponding author (email: bk_he@126.com)

(1) For a given grammar G=(V_N, V_T, S, P), indexing production rules in P in succession and constructing a table of pairs of the form (rule no. or nonterminal symbol, number of choices of the appropriate rule).

(2) Initializing the developing program (a sentential form) and genotypes as the start symbol S and sequences of codons respectively. Note that each codon here is represented by 8 bits.

(3) Evolving genotypes, via executing the following steps for evolved genotype repeatedly unless the developing program becomes a sentence of the chosen grammar (or some terminal condition is satisfied).

(a) Reading a codon of 8 bits from the concerned genotype and transforming it into an integer value.

(b) Choosing a production rule for the leftmost nonterminal symbol, say X, in the developing program based on the following mapping in order to make the leftmost derivation: *rule*= (*codon integer value*) *mod* (*number of rules of X*).

(c) Making the leftmost derivation using the rule obtained in b) for the developing program.

(d) Back to (a).

Therefore GE can be used to evolve programs in an arbitrary language within the constraints of what can be represented by a context-free grammar. It also has advantages over other GP variants [7–9, 14–16] in dealing with such important program problems as types [17] and recursion [18, 19].

In this paper, the three major problems focused are those that hinder the effective development of CGE. Since "most of the literature on GE is based on the use of natural transactions (i.e. natural binary encoding) and modulo translation" [20], i.e. based on CGE, this kind of problems is also common to many other GEs. The problems neglected in most existing GE researches are: i) how to structure the internal relationship among grammatical concepts, production rules and derivations; ii) how to improve the genotypic readability; and iii) how to reduce the search space so that it can possibly be extended to cope with more complex problems. The rest of this paper is organized as follows. In section 2, we introduce some preliminary background, which mainly includes concepts and some related problems of GE. Sections 3 through 5 focus on the proposed approach, concerning experiments, discussion as well. Finally in section 6, we conclude our work and draw future work directions.

2 Background

2.1 Grammar and programming language

Definition 2.1 (Grammar). A grammar for some language is a 4-tuple $G = (V_N, V_T, S, P)$, where V_N, V_T, S, P are defined as follows:

V_N: a set of nonterminal symbols. Elements of it are usually denoted as upper case letters;

V_T: a set of terminal symbols. Elements of it are usually denoted as lower case letters;

S: a special nonterminal symbol in V_N, called the start symbol;

P: a set of program generation rules of the form $A \to \alpha$, called production rules. Notice that when A has many alternatives, we shall treat them separately in the following discussion as different production rules.

Definition 2.2 (Derivation). Given a grammar $G = (V_N, V_T, S, P)$ and two strings $\alpha A\beta, \alpha\gamma\beta \in (V_N \cup V_T)^*$, we call $\alpha\gamma\beta$ a direct derivation from $\alpha A\beta$, denoted $\alpha A\beta \Rightarrow \alpha\gamma\beta$, if $A \to \gamma \in P$. If a series of direct derivations satisfies $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_n$, we call α_n a derivation from α_1, denoted $\alpha_1 \overset{*}{\Rightarrow} \alpha_n$. Particularly, a derivation is also called a zero derivation if it involves no production rule in the deduction process.

Definition 2.3 ($L(1)$ derivation). Given a grammar $G = (V_N, V_T, S, P)$, a production rule $A \to \gamma \in P$, and a string $\delta = \alpha A\beta \in (V_N \cup V_T)^*$, a direct derivation $\alpha A\beta \Rightarrow \alpha\gamma\beta$ is a L (1) derivation with respect to $A \to \gamma$, denoted $\alpha A\beta \underset{l(1)}{\overset{A\to\gamma}{\Rightarrow}} \alpha\gamma\beta$, if it is a substitution of γ for the leftmost occurrence of some nonterminal symbol A in δ.

Definition 2.4 ($\delta_{\mathrm{A}}^{\gamma}$). Given a grammar $G = (V_N, V_T, S, P)$, a production rule $A \to \gamma \in P$, and a string

$\delta \in (V_N \cup V_T)^*, \delta_A^\gamma$ is a string obtained from the substitution of γ for the leftmost occurrence of some nonterminal symbol A in δ. Particularly, if $\delta \in V_T^*$, we define $\delta_A^\gamma = \delta$, and regarding $\delta \Rightarrow \delta_A^\gamma$ as zero derivation.

So, $\delta \Rightarrow \delta_A^\gamma$ is either an L (1) derivation, if δ contains the nonterminal symbol A, or a zero derivation (i.e. $\delta = \delta$).

Definition 2.5 (Sentential form). Given a grammar $G = (V_N, V_T, S, P), \alpha \in (V_N \cup V_T)^*$ is a sentential form of G, if $S \overset{*}{\Rightarrow} \alpha$. Particularly, α is also called a sentence of G if it consists of only terminal symbols from V_T.

Definition 2.6 (Leftmost derivation). Given a grammar $G = (V_N, V_T, S, P)$, a production rule $A \rightarrow \gamma \in P$, and a string $\delta = \alpha A \beta \in (V_N \cup V_T)^*$, a direct derivation $\alpha A \beta \Rightarrow \alpha \gamma \beta$ is the leftmost derivation, denoted $\alpha A \beta \underset{lm}{\Rightarrow} \alpha \gamma \beta$, if it is a substitution of γ for the leftmost nonterminal symbol, say, A in δ.

Definition 2.7 ($L(1)$ sentential form). Given a grammar $G = (V_N, V_T, S, P)$, a sentential form α is of $L(1)$, if each direct derivation involved for deducing α from S is an $L(1)$ derivation.

Definition 2.8 (LM sentential form). Given a grammar $G = (V_N, V_T, S, P)$, a sentential form α is of LM, if each direct derivation involved for deducing α from S is a leftmost derivation.

Definition 2.9 (Language). Given a grammar $G = (V_N, V_T, S, P)$, the language $L(G)$ of G is $\{\alpha | S \overset{*}{\Rightarrow} \alpha \in V_T^*\}$.

Definition 2.10 ($L1(G)$). Given a grammar $G = (V_N, V_T, S, P)$, the $L(1)$ language of G, denoted $L1(G)$, is $\{\alpha \in L(G) | \alpha$ is an $L(1)$ sentential form$\}$.

Definition 2.11 ($LM(G)$). Given a grammar $G = (V_N, V_T, S, P)$, the leftmost language of G, denoted $LM(G)$, is $\{\alpha \in L(G) | \alpha$ is an LM sentential form$\}$.

Note that, $LM(G)$, the language defined by the leftmost derivations (commonly used in compiler construction), is a subset of $L1(G)$. For more details about grammar and language, refer to standard texts in the area such as [5, 6].

2.2 Problems of GE

O'Neill and Ryan [1] stated that "GE does not suffer from the problem of having to ignore codon integer values because it does not generate illegal values." Furthermore, to ensure a complete mapping of an individual onto a program comprised exclusively of terminals, they employed a novel technique called wrapping to partly guarantee this requirement. Overall, this technique works well. Nonetheless, equally important are such issues as space reduction, genotypic readability, concise representation, analytical study, and effective evaluation, etc. They have remained untouched since the advent of GE, and problems with them can have a negative effect on GE performance.

For instance, let Table 1 be a description of some genotype along with some of its GE properties. The first line stands for a genotype whose codons (integers in [0, 255]) are referred to as a_i; the second row lists rules determined by the GE mapping for the leftmost derivations; the third line shows the corresponding nonterminals for derivations to use, and the fourth row gives, for each involved nonterminal, the maximum choices of production rules. We can informally define search space as the set of all genotypes GE may search. It might seem that the whole search space is too large for the GE search algorithm to efficiently examine. In this case, the order of the search space is 256^n. Perhaps redundancy is helpful for the convergence of solutions, but it is at cost of efficiency, involving a considerable number of modulo arithmetic computations in a complex parse tree.

In section 3, we will tackle them by incorporating syntactically usable information into a finite state transition system. According to this method, both "codon" and that integer modulo based mapping become unnecessary. Search space is reduced to $\mathrm{O}(k^{n/m})$, where k stands for the maximum choices of certain production rule; n/m means, for a given genotype of length n, there are n/m genotypic components which have multiple choices. Building block can also be visually studied in the context of transition dia-

Table 1 A possible genotype with its GE properties

Genotype codons	$a_1 \in$[0, 255]	$a_2 \in$[0, 255]	$\cdots$	$a_{n-1} \in$[0, 255]	$a_n \in$[0, 255]
Derivation rules	$r_1 = a_1 \bmod m_1$	$r_2 = a_2 \bmod m_2$	$\cdots$	$r_{n-1} = a_{n-1} \bmod m_{n-1}$	$r_n = a_n \bmod m_n$
Nonterminals	N_1	N_2	$\cdots$	N_{n-1}	N_n
Max choices	m_1	m_2	$\cdots$	m_{n-1}	m_n

grams. Besides, the space reduction may benefit the present approach in the value of coping more complex problems.

3 Model for grammatical evolution

The following discussion focuses on the problem of modeling the language $LM(G)$ of some given context free grammar G.

3.1 Modeling principle

In [21, 22], we elaborate on our original study on "combining Hoare-logic-style assertion based specifications and model checking within a genetic programming framework" [23]. The method used there consists of three steps: i) Defining the search space as a set of Hoare triples; ii) modeling search space using finite state transition systems; iii) searching over some transition system for the desired Hoare triple.

In the following subsections, we deal with grammatical evolution in similar approach: i) Defining a programming language; ii) modeling grammatical derivations of the concerned language using finite state transition systems; iii) heuristically search over some obtained transition system for the sequence of production rules from which the desired program can grammatically be derived. Note that the major problem among these steps is how to construct the transition system. Technically, we first represent states of the transition diagram as certain sets of sentential forms of some grammar; then decide under some derivation convention, say the leftmost derivation, which states can be connected to which others.

3.2 Model and existence theorem

Definition 3.1 (LM justification)**.** Given a grammar $G = (V_N, V_T, S, P)$ and two sentential forms α and β, a sequence $s = p_1p_2\cdots p_n$ of production rule names is an LM justification of $\alpha \overset{*}{\Rightarrow} \beta$, if α, β satisfy $\alpha \underset{lm}{\overset{p_1}{\Rightarrow}} \alpha_1 \underset{lm}{\overset{p_2}{\Rightarrow}} \cdots \underset{lm}{\overset{p_{n-1}}{\Rightarrow}} \alpha_{n-1} \underset{lm}{\overset{p_n}{\Rightarrow}} \beta$.

Definition 3.2 (Grammar graph)**.** Given a grammar $G = (V_N, V_T, S, P)$, a finite state transition graph $Gph = \langle V, E\rangle$ is called a grammar graph, if each vertex in *Gph* is a set of sentential forms, and each edge in *Gph* is labeled either by a production rule (name) or an empty word ε. Particularly, if a string $e_1e_2\ldots e_{n-1}$ concatenated from edge labels along a path, say $V_1e_1V_2e_2\ldots e_{n-1}V_n$, forms an LM justification of $\alpha_1 \overset{*}{\Rightarrow} \alpha_n$ (where $\alpha_i \in V_i$, $1 \leqslant i \leqslant n$), it is also called an LM route of α_n from α_1.

Definition 3.3 (LM grammar model)**.** Given a grammar $G = (V_N, V_T, S, P)$ and a finite state transition graph $Gph = \langle V, E\rangle$ as above, the graph *Gph* is an LM grammar model of the grammar G, denoted $LMGM(G)$, if for $\alpha \in (V_N \cup V_T)^*$, α is a leftmost sentential form of $G \Leftrightarrow$there exists an LM route of α from start symbol S (a special sentential form of G) in a special vertex, called the initial vertex, of *Gph*. Particularly, it is true for the cases of sentences of $LM(G)$, too.

Theorem 1. Given a grammar $G = (V_N, V_T, S, P)$, there exists a grammar model $LMGM(G)$ for G.

Proof. Vertices in $LMGM(G)$, which is constructed by the following algorithm, can be put into two kinds. One of these kinds is of the form S_N, and the other one is S_N^α where $N \in V_N$.

Algorithm 1.

(1) Draw two vertices S_N and S_N^α for each production rule $N \to \alpha \in P$ (when N has many alternatives, we should treat them separately as different production rules), where $S_N = \{\alpha | \alpha$ is a sentence of G or

a sentential form of G whose leftmost nonterminal symbol is $N\}$, and $S_N^\alpha = \{\beta_N^\alpha | \beta \in S_N$ and β_N^α is the string obtained from substitution of α for the leftmost nonterminal symbol N in $\beta\}$.

(2) Draw an ε arrow from V to V for each vertex V of step 1.

(3) Draw an arrow from S_N to S_N^α if $N \to \alpha \in P$ is a production rule, and labeling it with either the production rule or the rule name.

(4) Draw an ε arrow from S_M^α to S_N for each pair of vertices (S_M^α, S_N), if S_M^α, S_N have sentential forms whose leftmost nonterminal symbols are N.

The obtained graph is what we need, $LMGM(G)$. Here S_S is the initial vertex, ε means zero derivation.

Now for $\alpha \in (V_N \cup V_T)^*$, we prove that: α is an LM sentential form of $G \Leftrightarrow$ there exists an LM route of α in $LMGM(G)$ from start symbol S(a special sentential form of G) in $S_S = \{\alpha | \alpha$ is either a sentential form of G with the leftmost nonterminal symbol S or a sentence of $G\}$.

=>: Since α is an LM sentential form of G, there must exist a sequence of LM derivations: $S \underset{lm}{\Rightarrow} \alpha_1 \underset{lm}{\Rightarrow} \alpha_2 \underset{lm}{\Rightarrow} \cdots \underset{lm}{\Rightarrow} \alpha_n \underset{lm}{\Rightarrow} \alpha$. So the proof goes by induction on derivation steps.

(i) Induction base: For derivations $S = S$ (zero derivation) and $S \underset{lm}{\Rightarrow} \alpha_1$ ($S \to \alpha_1 \in P$), we have paths $S_S \xrightarrow{\varepsilon} S_S$ and $S_S \xrightarrow{S \to \alpha_1} S_S^{\alpha_1}$ in $LMGM(G)$ to justify the LM route of S or α_1 from S in S_S.

(ii) Induction step: Without loss of generality, supposing $S \underset{lm}{\Rightarrow} \alpha_1 \underset{lm}{\Rightarrow} \alpha_2 \underset{lm}{\Rightarrow} \cdots \underset{lm}{\Rightarrow} \alpha_n \underset{lm}{\Rightarrow} \alpha$ is a sequence of LM derivations, and the production rule used in $\alpha_n \underset{lm}{\Rightarrow} \alpha$ is $p : N \to \beta \in P$. Again supposing by induction hypothesis that there exists an LM route $S_S \xrightarrow{p1} V_1 \xrightarrow{p2} V_2 \cdots \xrightarrow{pn} V_n$ of α_n ($\in V_n$) in $LMGM(G)$ from start symbol $S \in S_S$ for justification of $S \underset{lm}{\Rightarrow} \alpha_1 \underset{lm}{\Rightarrow} \alpha_2 \underset{lm}{\Rightarrow} \cdots \underset{lm}{\Rightarrow} \alpha_n$. We manage to prove that there must exist a path $S_S \xrightarrow{p1} V_1 \xrightarrow{p2} V_2 \cdots \xrightarrow{pn} V_n \xrightarrow{\varepsilon} S_N \xrightarrow{p} S_N^\beta$ in $LMGM(G)$ such that $p_1p_2 \cdots p_n\varepsilon p$ forms an LM route of α in S_N^β from $S \in S_S$. The proof is as follows.

Since the LM derivation $\alpha_n \underset{lm}{\Rightarrow} \alpha$ is obtained from applying $p : N \to \beta \in P$ to α_n, by Definition 3.3 the leftmost nonterminal symbol of α_n must be N. By the convention of S_N in step 1 of Algorithm 1, we have: $\alpha_n \in S_N$ and α_n is not a sentence. Now the proof goes on two cases.

Case 1 ($\alpha_n \in V_n = S_M$): From $\alpha_n \in V_n = S_M$ and the obtained result: $\alpha_n \in S_N$ and α_n is not a sentence, it follows $N = M$ (i.e. S_M is S_N). According to the before-mentioned construction method of $LMGM(G)$, there is an ε arrow leading from $S_M(= S_N)$ to S_N, that is, $S_S \xrightarrow{p1} V_1 \xrightarrow{p2} V_2 \cdots \xrightarrow{pn} V_n(= S_M = S_N) \xrightarrow{\varepsilon} S_N \xrightarrow{p} S_N^\beta$ in $LMGM(G)$.

Case 2 ($\alpha_n \in V_n = S_M^\gamma$): From $\alpha_n \in V_n = S_M^\gamma$ and the obtained result: $\alpha_n \in S_N$ and α_n is not a sentence, we can deduce S_M^r and S_N are states containing the same sentential form α_n with the leftmost nonterminal symbol N. According to step 4 of Algorithm 1, we have an ε arrow leading from S_M^γ to S_N, i.e. $S_S \xrightarrow{p1} V_1 \xrightarrow{p2} V_2 \cdots V_{n-1} \xrightarrow{pn} V_n(= S_M^r) \xrightarrow{\varepsilon} S_N \xrightarrow{p} S_N^\beta$ in $LMGM(G)$.

Obviously, $p_1p_2 \cdots p_n\varepsilon p$ of either case 1 or 2 forms an LM route of α (in S_N^β) from $S \in S_S$. This is what we need.

<=: By definitions and induction on length of routes, it is easy to demonstrate.

This completes the proof of Theorem 1.

Note that step 4 of Algorithm 1 technically relies on the computation of the so called leftmost connection function of subsection 3.3.

3.3 Modeling algorithm

This subsection concerns one decision problem on which the existence theorem strongly relies. For instance, step 4 of Algorithm 1 depends on Problem 1.

Problem 1. Given a grammar $G = (V_N, V_T, S, P)$ and a nonterminal symbol $A \in V_N$. Decide for S_M^α in $LMGM(G)$ whether it includes a sentential form taking A as the leftmost nonterminal symbol.

In fact, if we have certainty that there exists some sentential form in S_M^α taking A as the leftmost nonterminal symbol, we can then draw an ε arrow from vertex S_M^α to S_A.

Definition 3.4 (Leftmost connection function)**.** The leftmost connection function LMC of vertices of the form S_M^α in $LMGM(G)$ is defined as $LMC(S_M^\alpha) = \{A \in V_N | A$ occurs as the leftmost nonterminal symbol in some sentential form of $S_M^\alpha\}$.

So, to solve Problem 1, we should manage to solve the connection function. In fact, step 4 of Algorithm 1 can be revised as: for each pair of vertices (S_M^α, S_N) with $LMC(S_M^\alpha) \cap \{N\} \neq \emptyset$, draw an ε arrow from S_M^α to S_N.

Now, let us deal with Problem 1. The $FOLLOW$ function given below is used to construct the leftmost connection function of Definition 3.4.

Definition 3.5 ($FOLLOW$ function)**.** Given a grammar $G = (V_N, V_T, S, P)$, the $FOLLOW$ function of nonterminal symbols is a mapping $V_N \rightarrow 2^{V_N}$ such that:

(i) $Y \in FOLLOW(X)$, if there exists a production rule $A \rightarrow \cdots X\alpha Y \cdots \in P$ with $\alpha \in V_T^*$, and X, $Y \in V_N$;

(ii) $FOLLOW(A) \subseteq FOLLOW(X)$, if $A \rightarrow \cdots X\alpha \in P$ with A, $X \in V_N$, $\alpha \in V_T^*$.

Algorithm 2. The leftmost connection function LMC of vertices of the form S_M^α in $LMGM(G)$ can be solved as follows:

$$LMC(S_M^\alpha) = \begin{cases} \{A\} & \alpha \notin V_T^* \text{ and } A \text{ is the leftmost nonterminal symbol of } \alpha, \\ FOLLOW(M) & \alpha \in V_T^*. \end{cases}$$

At last, we should point out that $LMGM(G)$ can be simplified against (leftmost) connection function. For example, if $LMC(S_M^\alpha) = LMC(S_M^\beta)$, then S_M^α and S_M^β can be combined into a unified state $(S_M^\alpha \cup S_M^\beta)$. This property is also reflected in section 4.

3.4 MGE: Model-based grammatical evolution

Unlike GE which regards genotypes as unreadable sequences of codons, model-based grammatical evolution (MGE) which is developed under grammar model of some grammar generates computer programs directly from genotypes comprised of production rule names. Regarding genetic operators of MGE, their implementation strategies are almost the same as those of GE, except for the fact that a grammar model is employed in this case to reduce search space and to guarantee the validity of operations and genetic operators are applied only on valid genotypic sequences of production rules.

In principle, MGE generates computer programs in any language as follows:

(1) Constructing the grammar model, say $LMGM(G)$, as done above for some given grammar G.

(2) Initializing both the developing program (a sentential form of G) and genotypes as the start symbol S and sequences of production rule names *consistent* with the concerned grammar model respectively.

(3) Evolving genotype and executing the following steps for some evolved genotype (a sequence of production rule names) repeatedly unless the developing program becomes a sentence of the concerned grammar or some terminal condition is satisfied.

(a) Reading a rule name from the genotype.

(b) Making the leftmost derivation for the developing program using the rule of (a).

(c) Back to (a) or executing (d) for some condition, say, whether the end of the concerned genotype has been arrived at.

(d) Replacing all nonterminal symbols in the developing program with predefined strings of terminal symbols.

So, in case of MGE, both search space and unreadability of genotypes of GE are effectively under control. For this one can further refer to section 4 and section 5. In addition, concepts like codon and the integer modulo algorithm which play critical roles in GE can thoroughly be omitted. Finally, since a grammar model covers all possibly valid derivations of sentential forms of the concerned grammar, MGE and GE are equivalent in functionality to each other. This is reflected in Theorem 2. Step (d) is also referred to as a "complete mapping principle".

Theorem 2. Let $G = (V_N, V_T, S, P)$ be a grammar and $\alpha \in (V_N \cup V_T)*$. GE deduces α under $G \Longleftrightarrow$ MGE can deduce it too.

4 Experiments

In this section, we demonstrate MGE through the symbolic regression problem given in [1, 8]. The particular functions examined are

Experiment 1: $f(y) = y^4 + y^3 + y^2 + y$,

Experiment 2: $g(y) = \sin(y^4 + y^2)$,

Experiment 3: $h(y) = \sin(\exp(\sin(\exp(\sin(y)))))$,

with a list of 20 input values like $\{-1, -0.9, -0.8, -0.76, -0.72, -0.68, -0.64, -0.4, -0.2, 0, 0.2, 0.4, 0.63, 0.72, 0.81, 0.90, 0.93, 0.96, 0.99, 1\}$ in the range of $[-1, 1]$. The grammar used in this problem is $G = (V_N, V_T, S, P)$, where V_N={expr, op, pre_op, var}, V_T = {sin, cos, exp, log, +, −, *, /, y, 1.0, (,)}, $S = \langle\text{expr}\rangle$, and P are the following production rules.

1) $\langle\text{expr}\rangle ::= \langle\text{expr}\rangle\langle\text{op}\rangle\langle\text{expr}\rangle$	(1-1)	2) $\langle\text{op}\rangle ::= +$	(2-1)
$\vert(\langle\text{expr}\rangle\langle\text{op}\rangle\langle\text{expr}\rangle)$	(1-2)	$\vert -$	(2-2)
$\vert\langle\text{pre\_op}\rangle(\langle\text{expr}\rangle)$	(1-3)	$\vert *$	(2-3)
$\vert\langle\text{var}\rangle$	(1-4)	$\vert /$	(2-4)
3) $\langle\text{pre\_op}\rangle ::=$ sin	(3-1)	4) $\langle\text{var}\rangle ::= y$	(4-1)
$\vert$cos	(3-2)	$\vert$ 1.0	(4-2)
$\vert$exp	(3-3)		
$\vert$log	(3-4)		

Method. According to the existence theorem, we obtain a grammar model of Figure 1 within which only two edges have 4 choices. So for any n, the search space for a genotype of length n has the upper bound $O(4^{n/m})$. By n/m we mean, for the concerned genotype, there are n/m genotypic components embracing 2 or 4 choices. In light of this model, various variants of MGE using sequences of production rules as genotypes can then be developed. In this paper, a building block based MGE, called BGE, is now implemented in computer system. To better validate the effectiveness and efficiency of the proposed method, we will, in Figures 2–7, compare it not only with CGE, but also with some other GEs like IGE (integer-coded grammatical evolution) [24] and PIGE (or π GE) [25]. It follows that BGE can do the same as CGE, IGE, and PIGE in evolving programs in any language. The major parameters employed in the experiments are as follows:

Generation size: 100; Probability of crossover: 0.9; Crossover mode: two-point;

Population size: 50; Probability of mutation: 0.15; Mutation mode: block mutation;

Selection strategy: tournament; Runs: 100; Fitness evaluation: the least square error;

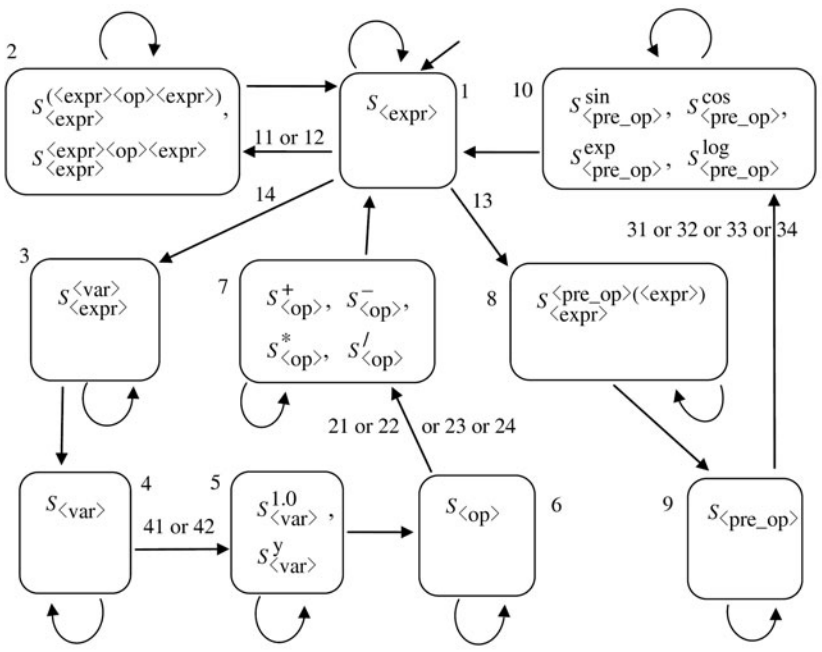
where the parameter Runs (=100) defines the number of runs that will be conducted for each experiment on a particular problem. The complete mapping $I : V_N \rightarrow V_T^*$ is

$$I(X) = \begin{cases} y & X \text{ is } \langle\text{expr}\rangle, \\ y & X \text{ is } \langle\text{var}\rangle, \\ + & X \text{ is } \langle\text{op}\rangle, \\ \sin & X \text{ is } \langle\text{pre\_op}\rangle. \end{cases} \quad \text{(Complete mapping)}$$

5 Discussion

Visualization has been well recognized as an important approach to the analysis of complex systems. What is the relationship among grammatical concepts of a given grammar? What is the essence of building blocks of GE? Particularly, what are the benefits these properties may give us in problem solving process? Questions like these occur frequently in genetic programming and are rarely touched in existing GE researches.

This work provides more means than GE does to partly understand or solve them. For instance, through the use of grammar model, say Figure 1, of some grammar, the relationship among nonterminal symbols, production rules and derivations are visualized graphically. Again from Figure 1, it follows that



1
$S_{\langle expr\rangle}$
2
$S_{\langle expr\rangle}^{(\langle expr\rangle\langle op\rangle\langle expr\rangle)}$, $S_{\langle expr\rangle}^{\langle expr\rangle\langle op\rangle\langle expr\rangle}$
11 or 12
10
$S_{\langle pre\_op\rangle}^{sin}$, $S_{\langle pre\_op\rangle}^{cos}$, $S_{\langle pre\_op\rangle}^{exp}$, $S_{\langle pre\_op\rangle}^{log}$
14
13
31 or 32 or 33 or 34
3
$S_{\langle expr\rangle}^{\langle var\rangle}$
7
$S_{\langle op\rangle}^{+}$, $S_{\langle op\rangle}^{-}$, $S_{\langle op\rangle}^{*}$, $S_{\langle op\rangle}^{/}$
8
$S_{\langle expr\rangle}^{\langle pre\_op\rangle(\langle expr\rangle)}$
21 or 22
or 23 or 24
4
$S_{\langle var\rangle}$
41 or 42
5
$S_{\langle var\rangle}^{1.0}$, $S_{\langle var\rangle}^{y}$
6
$S_{\langle op\rangle}$
9
$S_{\langle pre\_op\rangle}$

Figure 1 The leftmost grammar model $LMGM(G)$ of $G = (V_N, V_T, S, P)$. The arrows without labels stand for ε arrows. Labels represented by logic disjunctions can also be expressed as sets of rule names. For example, we can denote the rules 11 or 12 by {11, 12} (11 and 12 stand for rules 1-1 and 1-2 of the grammar). Node 1 is the start state of the automaton; the final state is technically omitted here.

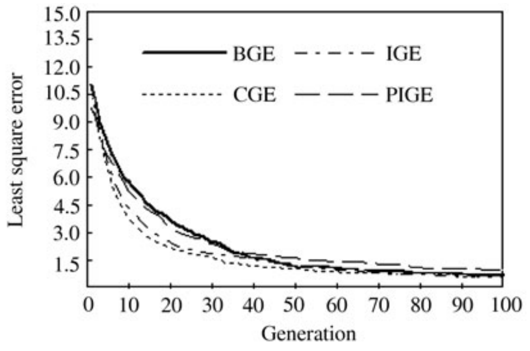


Figure 2 Average fitness of 100 runs of the four GEs in Experiment 1.

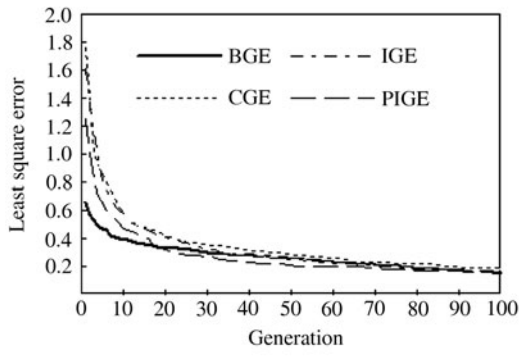


Figure 3 Average fitness of 100 runs of the four GEs in Experiment 2.

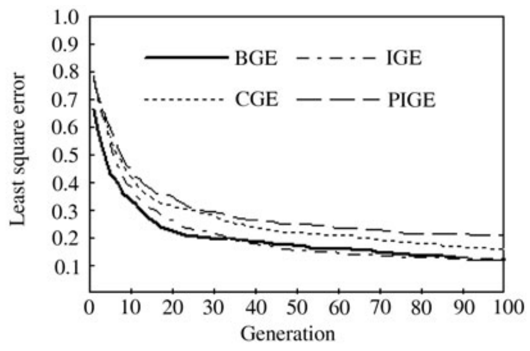


Figure 4 Average fitness of 100 runs of the four GEs in Experiment 3.

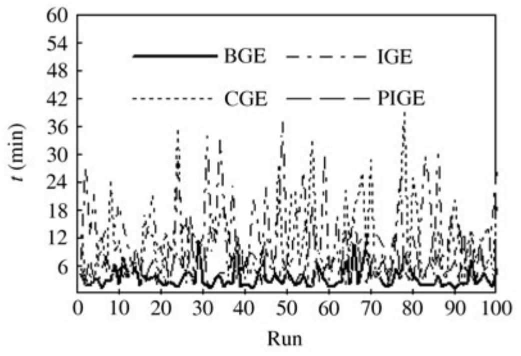


Figure 5 Time used of 100 individual runs of the four GEs in Experiment 1.

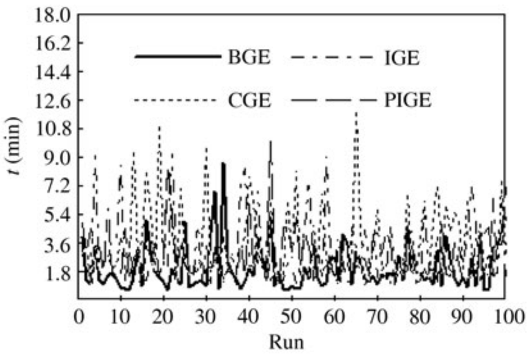


Figure 6 Time used of 100 individual runs of the four GEs in Experiment 2.

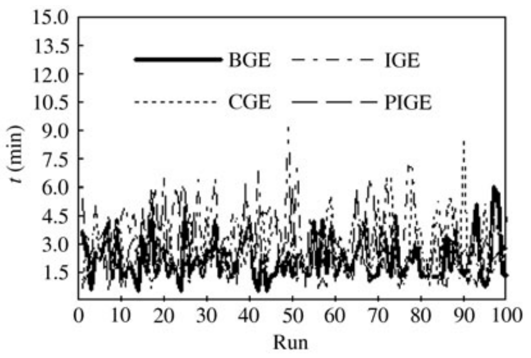


Figure 7 Time used of 100 individual runs of the four GEs in Experiment 3.

derivations of sentential forms of some given grammar can all be faithfully interpreted as sequences of such cycles as e, v and p, where e is the cycle $V_1V_2V_1$, v is $V_1V_3V_4V_5V_6V_7V_1$ and p is $V_1V_8V_9V_{10}V_1$. Naturally, we can treat e, v and p as building blocks, because any result of mutation or crossover of GE can essentially be represented by these cycles. So, the present approach makes it possible to develop more effective MGE (i.e. building block based GE) using building blocks. In this case, genotypes are elements (regular expressions) of $\{e, v, p\}^*$ and each component, say e, of such kind of genotypes will have definite semantics. Figures 2–7 are comparisons of 100 runs of the four GEs in solving $f(y), g(y)$, and $h(y)$. However, due to limited space, the details are omitted here. In short, MGE has advantages over GE in analytical study of grammatical evolution and effectiveness. Further experiments also give a positive outlook to this.

6 Conclusions

Model-based approach is an attractive method to achieve different objectives of software engineering. In this paper, we elaborate on our original study on grammatical evolution from model perspective, obtaining a novel GE called MGE. The major results are: 1) MGE is equivalent to GE in functionality and expressiveness; 2) MGE has advantage in revealing the relationship among grammatical concepts, production rules and derivations, contributing easiness for analytical studies in grammatical evolution; 3) possibly, MGE provides capability to solve more complex problems. Consequently, the combination of model-based approach and GE may be a novel direction towards effective program generations. Our future work includes: MGE tools, real world applications, and further comparisons of this approach with other GEs, etc.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (Grant Nos. 61170199, 60973053), the Engineering and Physical Sciences Research Council (EPSRC) (Grant No. EP/H020217/1), and the State Key Laboratory of Software Engineering of Wuhan University (Grant No. SKLSE 20080701). The authors are grateful to the reviewers for their helpful suggestions. In addition, He Pei would still like to give special thanks to the late Prof. Kang Lishan for introducing him to the area of evolutionary computation.

References

1 O'Neill M, Ryan C. Grammatical evolution. IEEE Trans Evolut Comput, 2001, 5: 349–358

2 Ryan C, Collins J J, O'Neill M. Grammatical evolution: Evolving programs for an arbitrary language. In: Banzhaf W, Poli R, Schoenauer M, et al., eds. Proc of the First European Workshop on Genetic Programming (EuroGP98), LNCS, 1998, 1391: 83–96

3 O'Neill M, Ryan C. Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language. Norwell, MA: Kluwer Academic Publishers, 2003

4 Mitchell M. An Introduction to Genetic Algorithms. Cambridge: MIT Press, 1996
5 Hopcroft J E, Motwani R, Ullman J D. Introduction to Automata Theory, Languages, and Computation. 3rd ed. San Antonio, TX: Pearson Education, Inc. 2008
6 Aho A V, Lam M S, Sethi R, et al. Compilers: Principles, Techniques, and Tools. 2nd ed. San Antonio, TX: Pearson Education, Inc. 2007
7 Koza J R. Genetic Programming. Cambridge MA: MIT Press, 1992
8 Oltean M, Grosan C. A comparison of several linear genetic programming techniques. Complex Syst, 2003, 14: 285–313
9 Sette S, Boullart L. Genetic programming: principles and applications. Eng Appl Artif Intell, 2001, 14: 727–736
10 Gavrilis D, Tsoulos I G, Dermatas E. Selecting and constructing features using grammatical evolution. Patt Recog Lett, 2008, 29: 1358–1365
11 Tsoulos I G, Gavrilis D, Glavas E. Neural network construction and training using grammatical evolution. Neurocomputing, 2008. 72: 269–277
12 Dempsey I, O'Neill M, Brabazon A. Adaptive trading with grammatical evolution. In: Proc of 2006 IEEE Congress on Evolutionary Computation. Vancouver, BC, Canada, 2006. 2587–2592
13 Tsoulos I G, Gavrilis D, Dermatas E. GDF: A tool for function estimation through grammatical evolution. Comput Phys Commun, 2006, 174: 555–559
14 Ferreira C. Gene expression programming: A new adaptive algorithm for solving problems. Complex Syst, 2001, 13: 87–129
15 Xu K K, Liu Y T, Tang R, et al. A novel method for real parameter optimization based on gene expression programming. Appl Soft Comput, 2009, 9: 725–737
16 Du X, Ding L X. About the convergence rates of a class of gene expression programming. Sci China Inf Sci, 2010, 53: 715–728
17 Pierce B C. Types and Programming Languages. Cambridge, MA: The MIT Press, 2002
18 Boolos G S, Burgess J P, Jeffrey R C. Computability and Logic. 4th ed. Cambridge: Cambridge Univ. Press, 2002
19 Wong M L, Mun T. Evolving recursive programs by using adaptive grammar based genetic programming. Genetic Program Evolv Mach, 2005, 6: 421–455
20 Wilson D, Kaur D. Search, neutral evolution, and mapping in evolutionary computing: A case study of grammatical evolution. IEEE Trans Evolut Comput, 2009, 13: 566–590
21 He P, Kang L S, Fu M. Formality based genetic programming. In: IEEE Congress on Evolutionary Computation. Hong Kong, 2008
22 He P, Kang L S, Johnson C G, et al. Hoare logic-based genetic programming. Sci China Inf Sci, 2011, 54: 623–637
23 Harman M, Mansouri S A, Zhang Y Y. Search based software engineering: A comprehensive analysis and review of trends techniques and application. Technical Report, TR-09-03, 2009
24 Hugosson J, Hemberg E, Brabazon A, et al. Genotype representations in grammatical evolution. Appl Soft Comput, 2010, 10: 36–43
25 O'Neill M, Brabazon A, Nicolau M, et al. πGrammatical evolution. In: Deb K, Poli R, Banzhaf W, et al. eds. Proc. GECCO, LNCS, 2004, 3103: 617–629