Soft Comput (2017) 21:5413–5423
DOI 10.1007/s00500-016-2130-1

METHODOLOGIES AND APPLICATION

Model approach to grammatical evolution: deep-structured analyzing of model and representation

Pei He[1,2,3] · **Zelin Deng**[2] · **Chongzhi Gao**[1] · **Xiuni Wang**[1] · **Jin Li**[1,4]

Published online: 12 April 2016

Abstract Grammatical evolution (GE) is a combination of genetic algorithm and context-free grammar, evolving programs for given problems by breeding candidate programs in the context of a grammar using genetic operations. As far as the representation is concerned, classical GE as well as most of its existing variants lacks awareness of both syntax and semantics, therefore having no potential for parallelism of various evaluation methods. To this end, we have proposed a novel approach called model-based grammatical evolution (MGE) in terms of grammar model (a finite state transition system) previously. It is proved, in the present paper, through theoretical analysis and experiments that semantic embedded syntax taking the form of regex (regular expression) over an alphabet of simple cycles and paths provides with potential for parallel evaluation of fitness, thereby making it possible for MGE to have a better performance in coping with more complex problems than most existing GEs.

Keywords Genetic programming · Grammatical evolution · Finite state automaton · Model

Communicated by V. Loia.

✉ Pei He
bk_he@126.com

Zelin Deng
zl_deng@sina.com

Chongzhi Gao
czgao@gzhu.edu.cn

Xiuni Wang
wang.xiuni@gmail.com

Jin Li
jinli71@gmail.com

[1] School of Computer Science and Educational Software, Guangzhou University, Guangzhou 510006, People's Republic of China

[2] School of Computer and Communication Engineering, Changsha University of Science and Technology, Changsha 410114, People's Republic of China

[3] Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin 541004, People's Republic of China

[4] Nanjing University of Information Science & Technology (NUIST), Nanjing 210024, People's Republic of China

1 Introduction

Solving complex problems by genetic programming (GP) was formally proposed and popularized by Koza (1992). This kind of work borrows ideas from genetic algorithm (GA), sharing the same algorithmic framework with GA, but provides more convenient means (Cano et al. 2015; Fernandez-Blanco et al. 2013; Fu et al. 2015; Kampouridis and Otero 2015; Krawiec 2014; Harman et al. 2012; Langdon and Harman 2015; O'Neill and Ryan 2001; Oltean et al. 2009; Qian et al. 2015) for dealing with nonlinear structures, and coping with a wide range of real-world optimization issues in circuit design, financial modeling, image processing, software engineering, etc.

Despite having achieved so many successes in GP applications, there still remain some important problems to be solved. In GP, solutions represented as abstract syntax trees (Aho et al. 2007; Hopcroft et al. 2008) are often evolved by randomly breeding candidate programs using genetic operators like crossover, mutation, and so on, thus giving rise to difficulties in delineating type, domain knowledge, restrictions on the legal programs, and closure requirement (when employing GP to generate computer programs, the first principle we should obey for valid crossovers or mutations is that we should guarantee type consistency). This principle is

referred to as closure (Koza 1992) in GP. To this end, various linear representation-based GP variants including particularly grammatical evolution (GE) (O'Neill and Ryan 2001) which forms our major concern in the present paper are developed. Owing to the use of a context-free grammar, GE as well as other grammar-related GP approaches can be best suited to type descriptions and recursive program generations.

Grammatical evolution is an automatic programming system, originally introduced by Ryan et al. (1998), and further described in later work by the same group. It differs from classical GP in ways of representation and decoding method. In GE (O'Neill and Ryan 2001), chromosomes usually represented by strings of codons (integers in [0,255], usually represented as 8 bits) are translated into sentential forms in light of the context-free grammar using a special genotype-to-phenotype mapping. Since GE outperforms GP in many aspects, such as easy for delineation of both type and domain knowledge, it has been widely taken up both by researchers and practitioners.

Works centered on improving GE, and debates on the effectiveness of the technique appear in a great many literatures (Alfonseca and Gil 2013; Burbidge and Wilson 2014; Dempsey et al. 2006; He et al. 2011b, 2015; Risco-Martin et al. 2014; Wilson and Kaur 2009; Mckay et al. 2010). For instance, O'Neill et al. (2004) extended classical GE by regarding the codon not as only one integer in [0, 255] for rule selection, but as a vector of pairs of integers in [0, 255] for designating both non-terminal and rule selection. So this genotype-to-phenotype mapping mechanism can take a terminal or non-terminal at any stage, thus allow greater control over the depth of programs. O'Neill and Ryan (2004), made the first attempt to construct a coevolutionary system of grammar and corresponding solutions, opening up an exciting research direction towards the development of more complex programs.

Grammatical evolution approaches have received a great deal of interest in recent years, but few of them deal with genotypic unreadability, deep-structured analyzing of representations; and few of them are willing to discard the seemingly omnipotent genotype-to-phenotype mapping used in O'Neill and Ryan (2001), thus many of them lack the potential for parallelism of fitness evaluations. Fortunately, model-based approach proposed in our previous work (He et al. 2008, 2011a,b, 2015) can help with these problems. In this paper, we will make a deep investigation into its normalized structural representation and parallelism that may be useful in the design of efficient GE system in asynchronous parallel computing environments. What we can conclude is chromosomes of mode-based grammatical evolution (MGE) (He et al. 2011b, 2015) can be represented by regex (regular expression) for languages over an alphabet of simple paths and cycles. More importantly, MGE still has the potential for parallelism of various evaluation methods.

2 Related works

2.1 Grammatical evolution

Grammatical evolution is an automatic programming technique pioneered by Ryan et al. (1998). It represents programs by sequences of codons (each codon consists of 8 bits), and decodes them into sentential forms of a context-free grammar using the following rule. Given a grammar as shown in even-5 parity problem of Sect. 4, an example of GE decoding process is given in Table 1.

Rule = (integer codon value) mod (number of derivation alternatives for current non-terminal X)

There is a vast bibliography focusing on GE related works up to date. Changes with either its codon or genotype-to-phenotype mapping will lead to different GE variants. For instance, representing codons of classical GE by integers or production rules, the integer representation GE (Hugosson et al. 2010) and MGE (He et al. 2011b, 2015) will be obtained. Since the important rule almost all existing GE obeyed in choosing production for the leftmost derivation is the use of natural transaction (i.e., natural binary encoding) and modulo translation (Wilson and Kaur, 2009), our discussion mainly focuses on classical GE. In view of the fact that most existing GEs have deficiencies in semantic analysis and

Table 1 Decoding of individual of even-5 parity problem (see Sect. 4 for the grammar)

Expression	Con	Num	R = Con mod Num	Rule
$\langle prog \rangle$	8	1	0(the first rule)	$\langle prog \rangle ::= \langle expr \rangle$
$\langle expr \rangle$	12	4	0(the first rule)	$\langle expr \rangle ::= \langle expr \rangle \langle op \rangle \langle expr \rangle$
$\langle expr \rangle \langle op \rangle \langle expr \rangle$	15	4	3(the 4th rule)	$\langle expr \rangle ::= \langle var \rangle$
$\langle var \rangle \langle op \rangle \langle expr \rangle$	16	5	1(the second rule)	$\langle var \rangle ::= d1$
$d1\ \langle op \rangle \langle expr \rangle$	12	3	0(the first rule)	$\langle op \rangle ::= or$
$d1\ or\ \langle expr \rangle$	19	4	3(the 4th rule)	$\langle expr \rangle ::= \langle var \rangle$
$d1\ or\ \langle var \rangle$	18	5	3(the 4th rule)	$\langle var \rangle ::= d3$
$d1\ or\ d3$				

genotypic readability and lack of awareness of both syntax and semantics, we carry out researches on the characteristics of MGE.

2.2 MGE: model-based grammatical evolution

Model-based grammatical evolution (MGE) (He et al. 2011b, 2015) is a relatively new GE variant developed over its grammar model. As we know, any codon of a GE chromosome has no definite semantics, therefore is difficult to figure out its roles. To this end, we have investigated relationships among production rules, delineating them with finite state transition system. This system is the so-called grammar model. For the readers convenience, we recall in this section some notions and properties of (He et al. 2011b, 2015).

Definition 1 (Context-free Grammar; Aho et al. 2007) A context-free grammar $G = (V_N, V_T, B, P)$ consists of four components: (a) a set V_N of non-terminal symbols or syntactical variables; (b) a set V_T of terminal symbols; (c) a designation of one of the non-terminals in V_N as the start symbol B; (d) a set P of productions applied for generation of programs. Each production is of the form $A \to \alpha$. When A has many alternatives, we shall treat them differently in the following discussion. Note that "context-free grammar" is also shortened to "grammar" in the later sections.

Definition 2 (Derivations) Let $G = (V_N, V_T, B, P)$ be a grammar with $A \to \gamma \in P$, and $\alpha A\beta \in (V_N \cup V_T)^*$. A *direct derivation* of $\alpha\gamma\beta$ from $\alpha A\beta$, denoted $\alpha A\beta \Rightarrow \alpha\gamma\beta$, is a substitution of γ for some A in $\alpha A\beta$. Particularly, we call $\alpha A\beta \Rightarrow \alpha\gamma\beta$ the leftmost derivation, denoted $\alpha A\beta \underset{lm}{\overset{A\to\gamma}{\Longrightarrow}} \alpha\gamma\beta$, if $\alpha \in V_T^*$. By the way, $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_n$ and $\alpha_1 \underset{lm}{\Longrightarrow} \alpha_2 \underset{lm}{\Longrightarrow} \cdots \underset{lm}{\Longrightarrow} \alpha_n$ are abbreviated to $\alpha_1 \overset{*}{\Longrightarrow} \alpha_n$ and $\alpha_1 \underset{lm}{\overset{*}{\Longrightarrow}} \alpha_n$, respectively. Derivations with no production involved are referred to as *zero derivations*.

Definition 3 (δ_A^γ) Let G=(V_N, V_T, B, P) be a grammar with $A \to \gamma \in P$, and $\delta \in (V_N \cup V_T)^*$. δ_A^γ is a substitution of γ for the leftmost occurrence of A in δ. Particularly, we define $\delta_A^\gamma = \delta$ for $A \notin \delta$, and regard $\delta_A^\gamma = \delta$ as a zero derivation.

Definition 4 (Sentential form) Given a grammar $G = (V_N, V_T, B, P)$ and a string $\alpha \in (V_N \cup V_T)^*$, α is a sentential form of G, if $B \overset{*}{\Longrightarrow} \alpha$ (α is also called a sentence if $B \overset{*}{\Longrightarrow} \alpha \in V_T^*$). α is an LM sentential form, provided all direct derivations involved in $B \overset{*}{\Longrightarrow} \alpha$ are leftmost ones.

Definition 5 (Justification) Let $G = (V_N, V_T, B, P)$ be a grammar, and $\alpha, \beta \in (V_N \cup V_T)^*$. A sequence $s = p_1 p_2 \ldots p_n$ of productions justifies the derivations $\alpha \overset{*}{\Longrightarrow} \beta$, if s,α,β, can establish the series of derivations $\alpha \underset{lm}{\overset{p_1}{\Longrightarrow}} \alpha_1 \underset{lm}{\overset{p_2}{\Longrightarrow}} \alpha_2 \cdots \underset{lm}{\overset{p_{n-1}}{\Longrightarrow}} \alpha_{n-1} \underset{lm}{\overset{p_n}{\Longrightarrow}} \beta$ or $\alpha \underset{lm}{\overset{s}{\Longrightarrow}} \beta$.

Definition 6 (ϵ-equivalence) Let $G = (V_N, V_T, B, P)$ be a grammar, and $\alpha \in (V_N \cup V_T)^*$. Two justifications j_1, j_2 are ϵ-equivalent, if they are exactly the same except for usage of ϵ(empty word).

Definition 7 (Leftmost grammar model) Let $G = (V_N, V_T, B, P)$ be a grammar. A finite state transition graph $Gph =< V, E >$ with edges in E labeled either by productions (or production names) or empty (or ϵ) words is a leftmost grammar model of G, denoted $LMGM(G)$, if for $\alpha \in (V_N \cup V_T)^*$, α is a leftmost sentential form of $G \Leftrightarrow$ there exists a path starting at the initial state in $LMGM(G)$ such that the sequence s concatenated from edge labels along it justifies α, i.e., $B \underset{lm}{\overset{s}{\Longrightarrow}} \alpha$.

Definition 8 (Language) Let $G = (V_N, V_T, B, P)$ be a grammar. The language commonly used in compiler constructions is $LM(G)$={$\alpha \in V_T^* \mid \alpha$ is an LM sentential form of G}.

Theorem 1 (Existence theorem) *Given a grammar $G = (V_N, V_T, B, P)$, there exists a leftmost grammar model LMGM(G) constructed by Algorithm 1.*

Algorithm 1 (Construction of $LMGM(G)$)

Input: a grammar $G = (V_N, V_T, B, P)$.

Output: $LMGM(G)$. Here S_B, ε stand for the initial vertex and zero derivation, respectively.

1) Draw two vertices S_N and S_N^α for each production $N \to \alpha \in P$. When N has many alternatives, we should treat them separately as different production.

2) Draw an ε arrow from V to itself for each vertex V of step 1;

3) Draw an arrow from S_N to S_N^α for vertices of step 1 if $N \to \alpha \in P$; naming the arrow either with the production or the production name.

4) Calculate $Follow(X)$ for all X in V_N as follows:

A. $Y \in Follow(X)$, if there exists a production $A \to \cdots X\alpha Y \cdots \in P$ with $\alpha \in V_T^*$, and $X, Y \in V_N$;

B. $Follow(A) \subseteq Follow(X)$, if $A \to \cdots X\alpha \in P$.

Where $A, X \in V_N, \alpha \in V_T^*$.

5) Calculate $LMC(S_M^\alpha)$ for all states of the form S_M^α as follows:

A. $LMC(S_M^\alpha) = \{A\}$, if $\alpha \notin V_T^*$ and A is the leftmost non-terminal symbol of α ;

B. $LMC(S_M^\alpha) = Follow(M)$, if $\alpha \in V_T^*$.

6) Draw an ε arrow from S_M^α to S_N, for every state S_N with subscript $N \in LMC(S_M^\alpha)$.

The transition diagram obtained above reflects the leftmost derivation relations. We also call it the leftmost grammar model, denoted $LMGM(G)$. Since the number of its nodes has linear relationship with that of productions, the node complexity is O(|P|). Based on this diagram, *Model based grammatical evolution* (MGE) which generates computer programs evolutionarily from genotypes

comprising production names can be developed. However, for simplicity we do not intended to show MGE algorithm and related genetic operations here. In fact, MGE and classical GE share the same algorithmic framework except that they work on different representations. So the reader can find about them either in classical works (Koza 1992; O'Neill and Ryan 2001) or in He et al. (2011b, 2015). In the following section, we will make deep investigation into some features most existing GEs do not cover, such as easiness in structured genotype analysis, modular representation, functional reuse, effective implementation, etc.

3 Deep-structured analysis of MGE

Previously we have introduced a novel GE framework (He et al. 2011b, 2015), discussed a wide range of related issues which include building block-based genotypic representation, evolutionary mechanism, semantic reuse, automatic detecting of reusable sub-expressions, and so on. In this section, we will make a systematic research on its normalized structural representation and parallelism that may be useful in the design of efficient GE system in asynchronous parallel computing environments.

3.1 Structured representation

Generally speaking, standard genetic programming has no built-in support for establishing a modular solution of a problem (Dostal 2013). As far as most of the existing GEs are concerned, even if there are the same gene segments, they do not mean the same semantics. However, we can expect more semantic results in the case of MGE. As proved later, not only can we represent chromosomes of MGE by modules like cycles and simple paths, but also we can evaluate them on modules of concern in parallel. Now let us define terminologies in light of grammar model as follows.

Definition 9 (Path) Let $G = (V_N, V_T, B, P)$, as defined in Sect. 2.2, be a context-free grammar, $LMGM(G) = \langle V, E\rangle$ the leftmost grammar model of G, a path p with starting and ending vertices x, y in $LMGM(G)$, denoted ${}_x p_y$, is a finite sequence of edges connecting a series of vertices. Particularly, if all vertices of the path are distinct, we call it a simple path. For simplicity ${}_x p_y$ is often abbreviated as p.

Definition 10 (Cycle) Let $G = (V_N, V_T, B, P)$ be a context-free grammar,$LMGM(G) = \langle V, E\rangle$ its leftmost grammar model, a path ${}_x p_y$ in $LMGM(G)$ is a cycle, if both of the starting and ending vertices x, y for the path are identical. Particularly, if a cycle consists of a simple path, we call it a simple cycle, denoted ${}_x p$, where x is the starting and ending vertex.

Definition 11 (Length) The length of a path p, denoted $|p|$, in a grammar model is the number of edges on it.

Definition 12 (Joinable paths) Two paths ${}_x p_y, {}_u q_v$, in some grammar model are joinable, denoted ${}_x pq_v$, if the ending and staring vertices of p and q are the same.

Definition 13 (Path covering) Let ${}_x p_y, {}_u q_v$ be two paths in a context-free grammar model, ${}_x p_y$ covers ${}_u q_v$, if (a) $x = u$; (b) there exists a path ${}_v z_y$ such that p is the join of q and z, i.e., $p = qz$.

Similarly, let us formulate paths in grammar model in regular expressions.

Definition 14 (Regular expression) Given a finite alphabet Σ, the regular expressions (regexes for short) as well as the corresponding regular sets are recursively defined as follows.

(a) Every $a \in \Sigma, \Phi, \epsilon$ are regexes, their corresponding regular sets, denoted $L(a)$, $L(\Phi)$ and $L(\epsilon)$ are $\{a\}$, Φ and $\{\epsilon\}$, respectively.
(b) Let x, y be regexes, $L(x)$ and $L(y)$ their corresponding regular sets. Then xy, $(x|y)$, x^* (or denoted $(x)^*$) are regexes, and $L(x)L(y)$, $L(x) \cup L(y)$, and $(L(x))^*$ are their corresponding regular sets.

Definition 15 (Valid regex) A regex E over a finite alphabet Σ of paths in some grammar model is a valid regex with respect to some vertices x and y, denoted ${}_x E_y$ (${}_x E$ for short when $x = y$), if it is recursively defined as follows:

(a) Either Φ or an empty word (an ϵ arrow going from vertex x to y);
(b) Every path in Σ (with starting and ending vertices x and y);
(c) If ${}_x M_u, {}_v N_y$ are valid regexes, then so is the concatenation ${}_x MN_y$, if $u = v$; the union ${}_x(M|N)_y$, if $v = x$ and $u = y$; and the star operation ${}_x(M^*)_x$(${}_x M^*$ for short), if $x = u$.

Definition 16 (Path covering regex) A path regex E is a path covering regex of some path p in a grammar model, if there exists at least one element in $L(E)$ which covers p.

Lemma 1 *Let $G = (V_N, V_T, B, P)$ be a context-free grammar, $LMGM(G)$ be its leftmost grammar model. Then, ${}_x p_y$ is a path in LMGM(G) $\Leftrightarrow$ there exists a path covering regex over an alphabet of simple cycles and paths covering it.*

Proof Assuming that $C = \{c|c$ is a simple cycle attainable from the vertex x in $LMGM(G)\}$, $cycles\_in(x)$= $\{v|v$ is a vertex in a simple cycle attainable from the vertex x in $LMGM(G)\}$, $sink(LMGM(G))$= $\{v|v$ is a vertex with zero outdegree in $LMGM(G)\}$, and L is the set of

```
typedef struct {
      int NumVertices;
      int NumEdges;
      int vertices[n];    //the number of vertices, counting from "1".
      int arcs[n][n];   //adjacent matrix
} TypeGraph;

int visited[n];     //vertex x (<=n) is visited <=> visited [x]= 1
int VerVisited[n],  NumVisited=0;    //number of vertices currently visited
int VerCount;     //number of vertices of a cycle

cycle_detecting(TypeGraph G, int v){
      mark v as a visited vertex by executing visited[v] = 1;
      record currently visited vertex v by executing VerVisited[ NumVisited++ ] = v;
      for (int i=1;  i<=G.NumVertices;  i++){
            if ((G.arcs[v][i] = 1) and (visited[i] =0)), then invoke DFS(G, i);
            else if ((G.arcs[v][i] = 1) and (visited[i] = 1)) {
                  store the number of visited vertices NumVisited – 1 up to now to VerCount;
                  count vertices of a cycle by executing while (VerVisited[VerCount] <> i) VerCount--;
                  output non-trivial simple cycle by executing the following if-statement
                  if (VerCount!= NumVisited-1) {
                        for (int q=VerCount; q<NumVisited; q++) cout<<VerVisited[q];
                        cout<<i<<endl ;  // VerVisited[VerCount], ..., i are vertex of a cycle
                  }  //// a cycle is a trivial one, if it includes only one empty edge
            }
      }
      mark v as a unvisited vertex so that it can be used to construct a new cycle by executing :
      visited[v] = 0;  NumVisited --;
}
```

Fig. 1 Simple cycle detecting algorithm

all simple paths from vertex x to every w in $cycles\_in(x) \cup sink(LMGM(G))$; from $w(\in cycles\_in(x))$ to h $(\in sink(G))$; and from w $(\in cycles\_in(x))$ to h $(\in cycles\_in(x))$ in $LMGM(G)$. The proof goes by induction on the length of paths.

(i) Induction base: for $|_x p_y| = 0$, the path covering regex is Φ. For $|_x p_y| = 1$, every simple cycle or simple path which starts at the vertex x and contains the only edge of p is desirable.

(ii) Induction step: let the path ${}_x p_y = ({}_x\alpha_z)({}_z\beta_y)$, and $|{}_x\alpha_z| = n$, $|{}_z\beta_y| = 1$. By induction hypothesis, there exists a path covering regex, say γ, over $(C \cup L)$ covering ${}_x\alpha_z$. Hence, ${}_x p_y$ as a path in $LMGM(G)$ is covered either by γ or $\gamma\delta$, where δ is in $\{e | e$ is a simple cycle or path with starting vertex z covering $\beta\}$. This completes the proof. □

Theorem 2 (Structured path-regex) *Let $G = (V_N, V_T, B, P)$ be a context-free grammar, $LMGM(G)$ be its leftmost grammar model .Then, p as a sequence of leftmost derivations of G is a path starting at vertex S_B in $LMGM(G)$ $\Leftrightarrow$ there exists a path covering regex over an alphabet of simple cycles and paths covering it.*

The proof follows from Lemma 1. It tells us every sequence of derivations of a sentential form can be structured represented by simple cycles and paths. To this end, a major task is to detect simple cycles automatically. The algorithm is given in Fig.1.

Algorithm 2 (Simple cycle detecting) Let $G = (V_N, V_T, B, P)$ be a context-free grammar, $LMGM(G)$ be its leftmost grammar model. Solving under the grammar model all the simple cycles by calling $circle\_detecting(G, 1)$ as shown in Fig.1.

3.2 Potential for parallelism of fitness evaluations

So far, we have established that individuals of MGE can be structured represented with modules like simple paths and cycles. What we should do next is to reveal its potential for parallelism of various evaluation methods. To find out both sub-expressions and structured individual decoding methods, let us first introduce several basic concepts.

Definition 17 (List) List is a data structure with the following properties.

(a) [] is an empty list;
(b) :: is a list operator with $a_1 :: [a_2, \ldots, a_n] = [a_1, \ldots, a_n]$, which means a insertion of a_1 into some list $[a_2, \ldots, a_n]$;

(c) @ is another list operator with $[a_1, \ldots, a_n]@[b_1, \cdots, b_m] = [a_1, \ldots, a_n, b_1, \ldots, b_m]$, which means a combination of two lists $[a_1, \ldots, a_n]$ and $[b_1, \ldots, b_m]$.

Algorithm 3 Let $G = (V_N, V_T, B, P)$ be a context-free grammar, $LMGM(G)$ be its leftmost grammar model, and $s = p_1 p_2 \cdots p_n$ as a sequence of productions be a path in $LMGM(G)$. Then, the decoding method $Der(s)$ of MGE essentially works as follows:

(a) $Der(s) = \alpha$, if s is a production $M \to \alpha$;
(b) $Der(s) = \alpha$, if $1 < k \leq n$, $Der(p_1 p_2 \cdots p_{k-1}) = \alpha \in (V_T)^*$;
(c) $Der(s) = \alpha\beta\gamma$, if $Der(p_1 p_2 \cdots p_{k-1}) = \alpha M \gamma \in (V_N \cup V_T)^*$, $\alpha \in (V_T)^*$, and p_k is $M \to \beta$.

Definition 18 (Fledged/unfledged phrase) Given $G = (V_N, V_T, B, P)$, $LMGM(G)$ as above, a string α is a phrase, if there exists some non-terminal M with $M \underset{lm}{\overset{*}{\Longrightarrow}} \alpha$. Particularly, α is a fledged phrase, if $\alpha \in (V_T)^*$; otherwise, α is a unfledged phrase.

Definition 19 (Normalized phrase representation) Given $G = (V_N, V_T, B, P)$ as above, a list of phrases $[e_1, e_2, \ldots, e_m]$ is a normalized phrase representation of $\alpha = p_1 p_2 \cdots p_n$, a sequence of leftmost derivations of G, if α is a symbolic combination of $Der^{-1}(e_1)$, $\ldots$, $Der^{-1}(e_m)$, i.e., $\alpha = Der^{-1}(e_1)Der^{-1}(e_2)\ldots Der^{-1}(e_m)$, where $Der^{-1}(e_i)$ with $Der(Der^{-1}(e_i)) = e_i$ is the sequence of productions (represented as a substring of α) applied to grammatically derive e_i in G.

Theorem 3 (Existence of normalized representation) *Let $G = (V_N, V_T, B, P)$ be a context-free grammar, $LMGM(G)$ be the corresponding leftmost grammar model. Then there exists uniquely a normalized phrase representation $R = [e_1, e_2, \ldots, e_m]$ for $\alpha = p_1 p_2 \cdots p_k$, a path in $LMGM(G)$ which represents a sequence of leftmost derivations, with one of the following properties.*

(a) *R is a list of fledged phrases;*
(b) *R is a list of both fledged and unfledged phrases, then e_m, in terms of leftmost derivations in G, is the only unfledged phrase in R.*

Proof Inducting on the length of $\alpha = p_1 p_2 \cdots p_k$, it follows the existence. Now, let us deal with the uniqueness.

Let $R_1 = [e_1, e_2, \ldots, e_{p-1}, e_p, \cdots, e_m]$, $R_2 = [e_1, e_2, \ldots, e_{p-1}, t_p, \ldots, t_n]$, be two different normalized phrase representations of α with $e_p \neq t_p (1 \leq p \leq min(m, n))$. Then $Der^{-1}(e_p) \neq Der^{-1}(t_p)$. Since the first production used for deriving e_p and t_p locate at the same place in α, it follows either $Der^{-1}(e_p) \subset Der^{-1}(t_p)$ or $Der^{-1}(e_p) \supset Der^{-1}(t_p)$. Without loss of generality, assuming $Der^{-1}(e_p) \subset Der^{-1}(t_p)$, by the definition of Algorithm 3, we have $Der(Der^{-1}(e_p)) = Der(Der^{-1}(t_p))$ if e_p is a fledged phrase. This is in contradiction with $e_p \neq t_p$. Otherwise, we have $\alpha = Der^{-1}(e_1)Der^{-1}(e_2)\cdots Der^{-1}(e_{p-1})Der^{-1}(t_p) \supset \alpha$, if e_p is a unfledged phrase (in this case, e_p should be the last element of R_1). We meet a contradiction again. This completes the proof.

Now, it is time to deal with the calculation of normalized phrase representations.

Algorithm 4 (Computing of normalized representation)

Given $G = (V_N, V_T, B, P)$, $LMGM(G)$ as above, the normalized phrase representation for a path $\alpha = p_1 p_2 \cdots p_k$ is recursively solved by the following function $Norm\_rep(\alpha)$.

A. $Norm\_rep(\alpha) = []$ $\quad if\ |\alpha| = 0$
B. $Norm\_rep(\alpha) = [Der(\alpha)]@Norm\_rep(copy(\alpha, |Der^{-1}(Der(\alpha))| + 1, |\alpha|))$ $\quad if\ |\alpha| \geq 1$

Where $|\alpha|$ stands for the number of productions in α, $copy(\alpha, |Der^{-1}(Der(\alpha))| + 1, |\alpha|)$ is a function solving the subsequence β of α such that α is the combination of $Der^{-1}(Der(\alpha))$ and β, i.e., $\alpha = Der^{-1}(Der(\alpha))\beta$. For the details of $Der^{-1}(e)$, one can refer to definition 19.

Clearly, not only can MGE be easily developed on building blocks like simple cycles and paths, but also its genotypes provide a potential for parallelism of various evaluation methods. We will demonstrate these advantages in Sect. 4.

4 Experiments

Previously, we have demonstrated the efficiency of MGE (He et al. 2011b, 2015) through using regression problems. In this section, we will certify the result with even-3, 4, and 5 parity problems. By even-k parity problem (Koza 1992), we mean a Boolean even-parity function of k Boolean arguments which returns T (True) if an even number of its arguments are T; and returns F (False), otherwise. The grammar used in the even-5 parity problem is as follows. The experiments show us again that MGE which is developed on the basis of building blocks such as simple paths and cycles has advantage over GE in performance improvements. As for the explanation, one can refer to our previous work (He et al. 2015) for the details. The major concern here will focus on the use of results of Sects. 2.2 and 3, and introduce the problem-solving process.

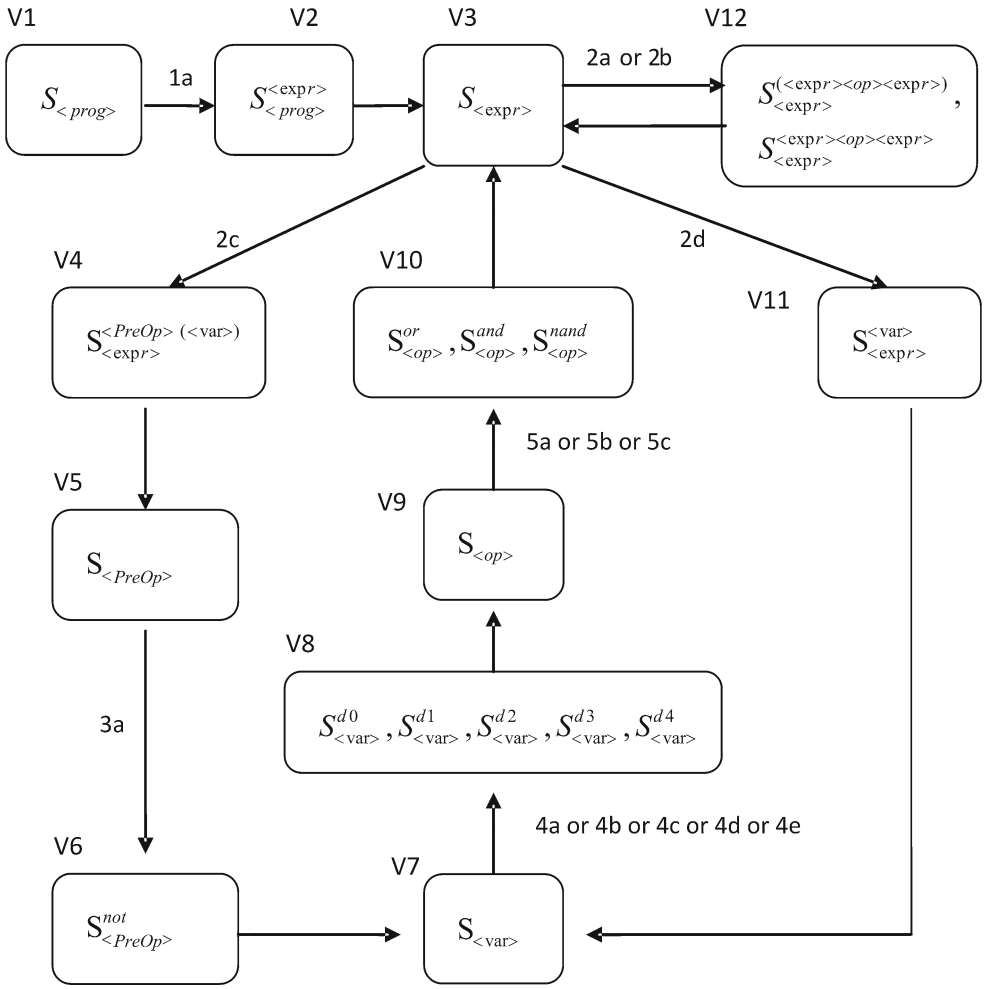

Fig. 2 The LMGM transition diagram for even-5 parity problem. Note that *arrows* without names are ϵ arrows. $S_{<prog>}$ is the start state. Besides, ϵ *arrows* over all states are omitted here

(1)	$\langle prog \rangle$	$::= \langle expr \rangle$	(1a)
(2)	$\langle expr \rangle$	$::= \langle expr \rangle \langle op \rangle \langle expr \rangle$	(2a)
		$\mid (\langle expr \rangle \langle op \rangle \langle expr \rangle)$	(2b)
		$\mid \langle PreOp \rangle (\langle var \rangle)$	(2c)
		$\mid \langle var \rangle$	(2d)
(3)	$\langle PreOp \rangle$	$::= not$	(3a)
(4)	$\langle var \rangle$	$::= d0$	(4a)
		$\mid d1$	(4b)
		$\mid d2$	(4c)
		$\mid d3$	(4d)
		$\mid d4$	(4e)
(5)	$\langle op \rangle$	$::= or$	(5a)
		$\mid and$	(5b)
		$\mid nand$	(5c)

The method solving even-parity problems is divided into 6 steps.

(a) Model construction: constructing by Theorem 1 the left-most grammar model of the grammar as shown in Fig. 2.

(b) Calculation of simple cycles: running Algorithm 2 on Fig. 2, we get three kinds of non-trivial cycles, denoted by e, v, and p. Here e, v, p are named after the first English letters of $[(] < expr >< op >< expr > [)]$, $< var >$ and $< PreOp > (< var >)$, respectively. Meanwhile, we identify nodes of Fig. 2 as $Vi (0 < i < 13)$, and represent these cycles as follows:

$$e : V3V12V3;$$
$$v : V3V4V5V6V7V8V9V10V3;$$
$$p : V3V11V7V8V9V10V3.$$

The adjacent matrix delineated by Fig. 2 is:

$$\begin{pmatrix}
1 & 1 & & & & & & & & & & \\
 & 1 & 1 & & & & & & & & & \\
 & & 1 & 1 & & & & & & & 1 & 1 \\
 & & & 1 & 1 & & & & & & & \\
 & & & & 1 & 1 & & & & & & \\
 & & & & & 1 & 1 & & & & & \\
 & & & & & & 1 & 1 & & & & \\
 & & & & & & & 1 & 1 & & & \\
 & & & & & & & & 1 & 1 & & \\
 & & 1 & & & & & & & 1 & & \\
 & & & & & & 1 & & & & 1 & \\
 & & 1 & & & & & & & & & 1
\end{pmatrix}$$

(c) Construction of structured path-regex: from Theorem 2 and results of steps a and b, it follows every chromosome can be represented by a structured path-regex, i.e., $(1a\epsilon)(e|v|p)^*$, where $1a\epsilon$ is the only simple path in $LMGM(G)$ (see Fig. 2).

(d) System implementation: developing building block based MGE in terms of the structured path-regex $(1a\epsilon)(e|v|p)^*$. However, to ensure all sentential forms generated by MGE can be successfully transformed into sentences (or programs), some default mapping principle, called pre-defined complete mapping function, should be employed. The involved pre-defined complete mapping function $I : V_N \rightarrow V_T^*$ used here is $I(X)$:

$$I(X) = \begin{cases} d0 & X\ is\ <expr> \\ d1 & X\ is\ <var> \\ or & X\ is\ <op> \\ not & X\ is\ <PreOp> \\ d0 & X\ is\ <prog> \end{cases}$$

(e) Parameter settings: The major parameters employed are as follows. The parameter $Runs(=100)$ defines the *number of runs* that will be conducted for each experiment on a particular problem. Fitness evaluation: the set of fitness cases for the problems of concern consist of 2^k combinations of the k Boolean arguments. So, the fitness is the sum, over these fitness cases, of the Hamming distance (error) between the returned values by MGE chromosome and the correct value of the Boolean function (Oltean 2004). The other parameters are as follows:
Generation size: 100
Probability of crossover: 0.9
Crossover model: two-point
Population size: 50
Probability of mutation: 0.15
Mutation mode: block mutation
Selection strategy: tournament
Runs: 100

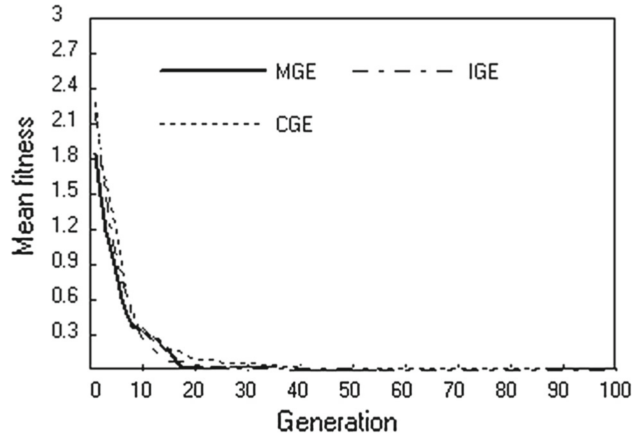

Fig. 3 Average fitness of 100 runs of the 3 GEs in even-3 parity problem

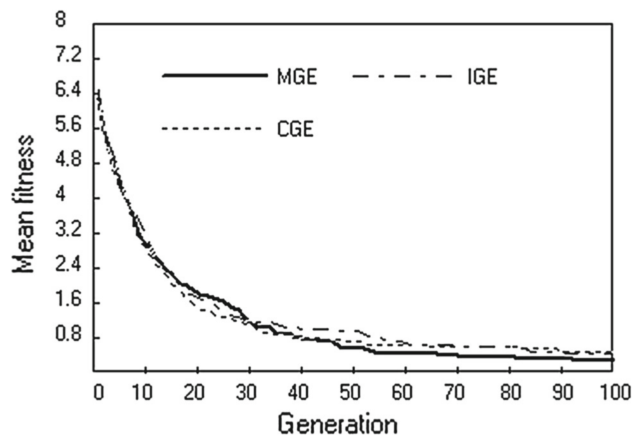

Fig. 4 Average fitness of 100 runs of the 3 GEs in even-4 parity problem

(f) Solving the problem: running all the involved 3 GEs with the above parameters on the given problems, we get the results shown in Figs. 3, 4, 5, 6, 7 and 8.

Figures 3, 4, 5 illustrate the average fitness of 100 runs of MGE, CGE, IGE in the experiments. Figures 6, 7, 8 compare these three methods with respect to time complexity. These figures, on the one hand, demonstrate that MGE has almost the same ability (refer to the shape of fitness profiles and the ultimate approximate solutions) as the other two GEs to generate the desired result, and the advantage in efficiency over the others. For an explanation, one can refer to He et al. (2015).

5 Discussion

While commenting on the open issues of GP in O'Neill et al. (2010), O'Neill et al. have pointed out that the most efficient forms of GP may combine awareness of both syntax and

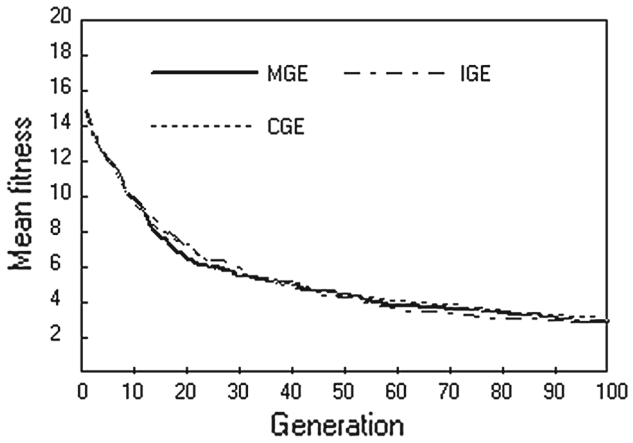

Fig. 5 Average fitness of 100 runs of the 3 GEs in even-5 parity problem

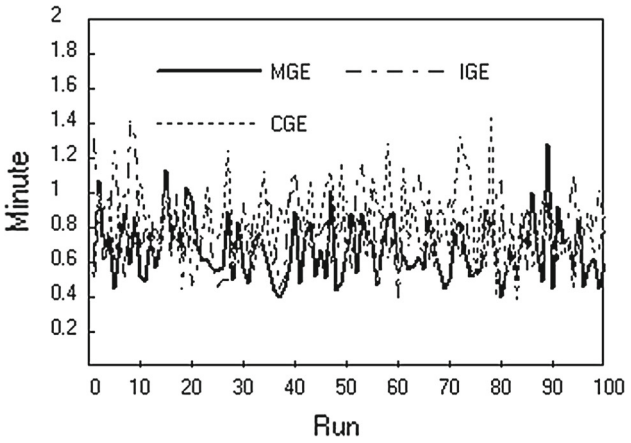

Fig. 8 Time used for 100 individual runs of the 3 GEs in even-5 parity problem

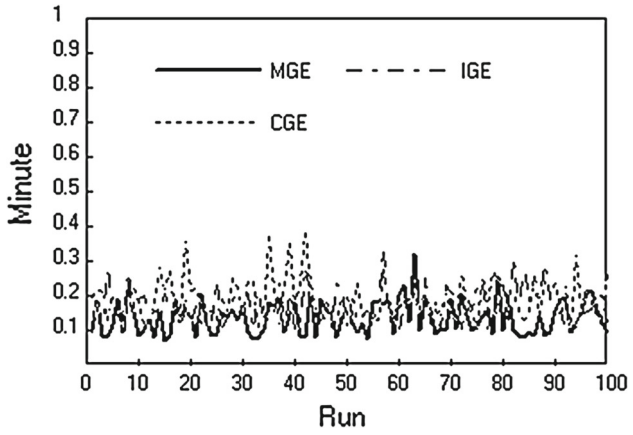

Fig. 6 Time used for 100 individual runs of the 3 GEs in even-3 parity problem

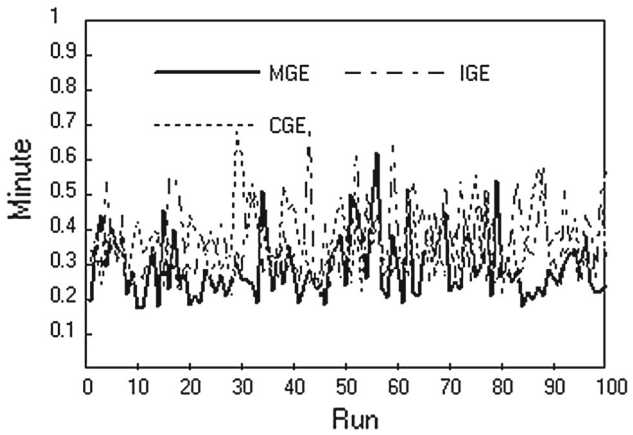

Fig. 7 Time used for 100 individual runs of the 3 GEs in even-4 parity problem

semantics. Unlike most of existing GEs which decode codons of individuals in terms of their context, MGE works only on sequences of productions. Since every production or segment of chromosomes has definite semantics, we have conducted modular analysis on phenotypes in He et al. (2015), and deep-structured representation in the present paper. Undoubtedly, these achievements benefit from representation and the awareness of both syntax and semantics.

Additionally, judging by the obtained results, we have good reason to benefit more from MGE. As we know, there exist many GP variants like GEP, MEP, etc., (Ferreira 2001; Du et al. 2015; Oltean et al. 2009). As far as expressiveness is concerned, we can construct them using context-free grammar. This furnishes a formal framework for unifying numerous GP variants. Consequently, many new methods introduced to improve GPs will find their way into MGE. Reusability has lain at the heart of software development issues for a long time. As revealed in He et al. (2015) and discussions above, reusable principle can be syntactically established on the basis of genotypic analysis technique of sub-expressions. Meanwhile, semantic embedded syntax taking the form of regex of simple cycles and paths supports parallel evaluations, and help to implement GE system efficiently. We will manage to find their new applications in a wide range of real-world optimization problems (Li et al. 2010, 2014; D'Apiec et al. 2014; Mokryani et al. 2013; Castiglione et al. 2015; Esposito et al. 2013; Gu et al. 2015; Ma et al. 2015; Wen et al. 2015; Xie and Wang 2014; Zheng et al. 2015) in the future.

Although structured analysis shows that chromosomes can be well expressed by regex, how to synthesize desirable expressions from grammar models of MGE is worthy of deep investigation. To this end, we may ask formal linguistic theory for the solution.

6 Conclusions

In place of representing genotype of GE by sequences of bits, model-based GE (MGE) which aims at improving genotypic

readability, reusability and fitness evaluations was developed over finite state transition systems as well as derivation paths in them (He et al. 2011b, 2015). The present paper recalls the major results, revealing deeply through theoretical studies and demonstrations why MGE supports structured representation, and why it has the potential for parallelism of fitness evaluations. So, MGE has many advantages over classical GE particularly in visualized analysis of chromosomes, semantic reuse, and performance improvement. Our future works will focus on its real-world applications, structure detecting, and unification with other GP variants.

Compliance with ethical standards

Funding This study was funded by National Natural Science Foundation of China (Grant Nos. 61170199, 61302061), the Natural Science Foundation of Guangdong Province, China (Grant No. 2015A030313501), the Scientific Research Fund of Education Department of Hunan Province, China (Grant No. 11A004), Science and Technology Program of Hunan Province, China (Grant No. 2015SK20463), the Priority Academic Program Development of Jiangsu Higer Education Institutions (PAPD), Jiangsu Collaborative Innovation Center on Atmospheric Environment and Equipment Technology (CICAEET), and the Open Fund of Guangxi Key Laboratory of Trusted Software (Guilin University of Electronic Technology) (Grant No. kx201208), Construction Project of Innovation Team in Universities in Guangdong Province, China (Grant No. 2015KCXTD014).

Conflicts of interest Pei He declares that he has no conflict of interest. Zelin Deng declares that he has no conflict of interest. Chongzhi Gao declares that he has no conflict of interest. Xiuni Wang declares that she has no conflict of interest. Jin Li declares that he has no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

Aho AV, Lam MS, Sethi R, Ullman JD (2007) Compilers: principles, techniques, and tools, 2nd edn. Pearson Education Inc., Upper Saddle River

Alfonseca M, Gil FJS (2013) Evolving an ecology of mathematical expressions with grammatical evolution. Biosystems 111(1):111–119

Burbidge R, Wilson MS (2014) Vector-valued function estimation by grammatical evolution. Inform Sci 258(1):182–199

Cano A, Ventura S, Krzysztof JC (2015) Multi-objective genetic programming for feature extraction and data visualization. Soft Comput. doi:10.1007/s00500-015-1907-y

Castiglione A, Pizzolante R, De Santis A, Carpentieri B, Castiglione A, Palmieri F (2015) Cloud-based adaptive compression and secure management services for 3D healthcare data. Future Gener Comput Syst 43–44:120–134

D'Apiec C, Nicola CD, Manzo V, Moccia R (2014) Optimal scheduling for aircraft departure. J Ambient Intell Hum Comput 5(1):799–807

Dempsey I, ONeill M, Brabazon A (2006) Adaptive trading with grammatical evolution. In Proc. of 2006 IEEE Congress on Evolutionary Computation, vol 1, pp 2587–2592

Dostal M (2013) Modularity in genetic programming. In: Zelinka I et al (eds) Handbook of optimization. ISRL, pp 38

Du X, Ni YC, Xie DT, Yao X, Ye P, Xiao RL (2015) The time complexity analysis of a class of gene expression programming. Soft Comput 19(6):1611–1625

Esposito C, Ficco M, Palmieri F, Castiglione A (2013) Interconnecting federated clouds by using publish-subscribe service. Clust Comput 16(4):887–903

Fernandez-Blanco E, Rivero D, Gestal M, Dorado J (2013) Classification of signals by means of genetic programming. Soft Comput 17(1):1929–1937

Ferreira C (2001) Gene expression programming: a new adaptive algorithm for solving problems. Complex Syst 13(2):87–129

Fu W, Johnston M, Zhang M (2015) Genetic programming for edge detection: a gaussian-based approach. Soft Comput 20(3):1231–1248

Gu B, Sheng VS, Tay KY, Romano W, Li S (2015) Incremental support vector learning for ordinal regression. IEEE Trans Neural Netw Learn Syst 26(7):1403–1416

Harman M, Mansouri SA, Zhang Y (2012) Search-based software engineering: trends, techniques and applications. ACM Comput Surv 45(1):1–61

He P, Kang LS, Fu M (2008) Formality based genetic programming. In: IEEE Congress on Evolutionary Computation

He P, Kang LS, Johnson CG, Ying S (2011a) Hoare logic-based genetic programming. Sci China Inform Sci 54(3):623–637

He P, Johnson CG, Wang HF (2011b) Modeling grammatical evolution by automaton. Sci China Inform Sci 54(12):2544–2553

He P, Deng ZL, Wang HF, Liu ZS (2015) Model approach to grammatical evolution: theory and case study. Soft Comput. doi:10.1007/s00500-015-1710-9

Hopcroft JE, Motwani R, Ullman JD (2008) Automata theory, languages, and computation, 3rd edn. Pearson Education Inc., Upper Saddle River

Hugosson J, Hemberg E, Brabazon A, O'Neill M (2010) Genotype representation in grammatical evolution. Appl Soft Comput 10(1):36–43

Kampouridis M, Otero FEB (2015) Heuristic procedures for improving the predictability of a genetic programming financial forecasting algorithm. Soft Comput. doi:10.1007/s00500-015-1614-8

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge

Krawiec K (2014) Genetic programming: where meaning emerges from program code. Genet Progr Evol Mach 15(1):75–77

Langdon WB, Harman M (2015) Optimizing existing software with genetic programming. IEEE Trans Evol Comput 19(1):118–135

Li J, Wang Q, Wang C, Cao N, Ren K, Lou WJ (2010) Fuzzy keyword search over encrypted data in cloud computing. In: Proceeding of the 29th IEEE International Conference on Computer Communications (INFOCOM 2010), pp 441–445

Li J, Huang XY, Li JW, Chen XF, Xiang Y (2014) Securely outsourcing attribute-based encryption with checkability. IEEE Trans Parallel Distrib Syst 25(8):2201–2210

Ma T, Zhou JJ, Tang M, Tian Y, AL-Dhelaan A, AL-Rodhaan M, Lee SY (2015) Social network and tag sources based augmenting collaborative recommender system. IEICE Trans Inform Syst E98–D(4):902–910

Mckay RI, Hoai NX, Whigham PA, Shan Y, ONeill M (2010) Grammar-based genetic programming: a survey. Genet Program Evol Mach 11(3/4):365–396

Mokryani G, Siano P, Piccolo A (2013) Optimal allocation of wind turbines in microgrids by using genetic algorithm. J Ambient Intell Hum Comput 4(1):613–619

Oltean M (2004) Solving even-parity problems using traceless genetic programming. In: IEEE Congress on Evolutionary Computation, vol 2.IEEE, pp 1813–1819

Oltean M, Grosan C, Diosan L, Mihaila C (2009) Genetic programming with linear representation: a survey. Int J Artif Intell Tools 19(2):197–239

O'Neill M, Ryan C (2001) Grammatical evolution. IEEE Trans Evol Comput 5(4):349–358

O'Neill M, Ryan C (2004) Grammatical evolution by grammatical evolution. In: Proceedings Of the 7th European Conference on genetic programming, vol 3003, LNCS, pp 138–149

O'Neill M, Brabzaon A, Nicolau M, Mc Garraghy S, Keenan P (2004) π grammatical evolution. In: Proc. GECCO, vol 3103, LNCS, pp 617–629

O'Neill M, Vanneschi L, Gustafson S, Banzhaf W (2010) Open issues in genetic programming. Genet Program Evol Mach 11(3):330–363

Qian C, Yu Y, Zhou Z-H (2015) Variable solution structure can be helpful in evolutionary optimization. Sci China Inform Sci 58(1):1–17

Risco-Martin JL, Colmenar JM, Hidalgo JI (2014) A methodology to automatically optimize dynamic memory managers applying grammatical evolution. J Syst Softw 91(1):109–123

Ryan C, Collins J, O'Neill M (1998) Grammatical evolution: evolving programs for an arbitrary language. In: Proceedings of the first European Workshop on Genetic Programming, vol 1391, LNCS, pp 83–96

Wen XZ, Shao L, Xue Y, Fang W (2015) A rapid learning algorithm for vehicle classification. Inform Sci 295(1):395–406

Wilson D, Kaur D (2009) Search, neutral evolution, and mapping in evolutionary computing: a case study of grammatical evolution. IEEE Trans Evol Comput 13(3):567

Xie SD, Wang YX (2014) Construction of tree network with limited delivery latency in homogeneous wireless sensor networks. Wirel Pers Commun 78(1):231–246

Zheng YH, Jeon BW, Xu DH, Wu QMJ, Zhang H (2015) Image segmentation by generalized hierarchical fuzzy c-means algorithm. J Intell Fuzzy Syst 28(2):961–973