Grammatical Evolution Using Tree Representation Learning

Shunya Maruta[1], Yi Zuo[2](✉), Masahiro Nagao[3], Hideyuki Sugiura[1], and Eisuke Kita[1]

[1] Graduate School of Information Sciences, Nagoya University, Nagoya, Japan
maruta.shunya@c.mbox.nagoya-u.ac.jp, sugiura.hideyuki@h.mbox.nagoya-u.ac.jp, kita@is.nagoya-u.ac.jp

[2] Institute of Innovation for Future Society, Nagoya University, Nagoya, Japan
zuo@nagoya-u.jp

[3] Graduate School of Environmental Science, Nagoya University, Nagoya, Japan
nagao@urban.env.nagoya-u.ac.jp

Abstract. Grammatical evolution (GE) is one of the evolutionary computations, which evolves genotype to map phenotype by using the Backus-Naur Form (BNF) syntax. GE has been widely employed to represent syntactic structure of a function or a program in order to satisfy the design objective. As the GE decoding process parses the genotype chromosome into array or list structures with left-order traversal, encoding process could change gene codons or orders after genetic operations. For improving this issue, this paper proposes a novel GE algorithm using tree representation learning (GETRL) and presents three contributions to the original GE, genetic algorithm (GA) and genetic programming (GP). Firstly, GETRL uses a tree-based structure to represent the functions and programs for practical problems. To be different from the traditional GA, GETRL adopts a genotype-to-phenotype encoding process, which transforms the genes structures for tree traversal. Secondly, a pointer allocation mechanism is introduced in this method, which allows the GETRL to pursue the genetic operations like typical GAs. To compare with the typical GP, however GETRL still generates a tree structure, our method adopts a phenotype-to-genotype decoding process, which allows the genetic operations be able to be apply into tree-based structure. Thirdly, due to each codon in GE has different expression meaning, genetic operations are quite different from GAs, in which all codons have the same meaning. In this study, we also suggest a multi-chromosome system and apply it into GETRL, which can prevent from overriding the codons for different objectives.

Keywords: Grammatical evolution · Tree representation · Multiple chromosomes · Pointer allocation · Genotype-phenotype map

1 Introduction

Genetic algorithm (GA) is one of the most popular algorithms in evolutionary computation [1,4,14], which has been considerably applied to optimization, adaptation and learning problems. However, some problems e.g., symbolic regression, syntactic problems and automatic generation program, are still hard to be solved for GAs to represent schema information. Therefore, genetic programming (GP) was proposed by Koza [5]. GP evolves a population of computer programs by using Lisp language to automatically solve problems without requiring the user to know or specify the form or structure of the solution in advance. Despite the advantages, GP also has several limitations. Recombination problem is the well-known one which limits its applicability and performance. First, sub-tree crossover in GP sometimes generated the invalid individual which was translated into incorrect function or program. Second, sub-tree crossover also had a tendency for parse trees to grow larger and larger, which would cause program size bloat.

Grammatical evolution (GE) is another tree-based evolutionary algorithm, which was presented by [2,8,10,12,13,15]. The main features of GE are to present an evolutionary process, map genotype to phenotype in a genetic algorithm approach, and translate rules using the Backus-Naur Form (BNF) syntax. Therefore, GE is very attractive in two folds: (1) The use of the translations rule can avoid the generation of the invalid phenotype. (2) The genotype-to-phenotype mapping can capture important schema information. In past few years, many works about the GE representations empirically measured the locality of GE, and identified that standard GE has low locality and compromises search effectiveness [3,9,11,16]. To enhance GE representations, several studies have sufficiently investigated grammar-guided GP and tree-based GE [6,9]. Murphy et al. [7] examined the behavior of tree-adjunct grammars to grammatical evolution. Whigham et al. [17] investigated the application of context-free grammar genetic programming. However, several limitations have revealed during the evolutionary process, such as, the genetic operators in GE overriding the original meaning of each codon in chromosome and violating the better partial structures of the phenotypes.

For these issues, this paper proposes a novel GE-based algorithm using tree representation learning (GETRL) and presents three contributions to the original GE and GP. First, GETRL uses a tree-based structure to represent the functions and programs for practical problems. To be different from the traditional GA, GETRL adopts a genotype-to-phenotype encoding process, which transforms the linear gene structures into tree traversal. In contrast to the original GE, which caused the overriding of codons when reading codons in array sequence with leftmost derivation, GETRL can represent translation process in advance to assign the codons as nodes of tree structure following the level-order traversal. Second, we introduce a pointer allocation mechanism to learn tree structures. During the translation process of genotype-to-phenotype mapping, a pointer is employed to map tree structure from the genotype chromosome. The pointer can link the nodes in virtually continuous address, which are distributed discretely in the linear genotype chromosome. When the sequence of chromosome is changed by genetic operators, the pointer remembers the address of the right position and finds the exact nodes in the parse tree. To compare with the typical

GP, the non-terminal nodes are labeled and linked to the corresponding codons in chromosome, which allows the GETRL to pursue the genetic operations by using the pointers. Third, we also introduce a multi-chromosomes system into GETRL, which can improve the strategy of genetic operators (e.g., crossover and mutation). Due to there are different kinds of nonterminals in BNF grammar definition, this paper uses multi-chromosomes each sub-chromosome is assigned to individual nonterminals to represent different objectives. Furthermore, the multi-chromosomes system can also prevent from overriding the codons for different objectives.

In Sect. 2, we describe the GETRL. In Sect. 3, we present the experiments, results and discussions. In Sect. 4, we conclude and summarize the paper.

2 Method

2.1 Tree Representation

The original GE used the binary strings to define individuals in genotype (Fig. 1). GE adopted a genotype-to-phenotype process, where individuals were denoted as integer-form using the grammar as shown in Table 1. From the leftmost unused integer number, the genotype was referred to g_i for $i = 0$ to n, and translation of genotype was started from symbol `<expr>` (see Fig. 2). We assumed that the leftmost untranslated symbol in the phenotype is α and the number of the substitution rules for α is s_α. When the remainder r_i was calculated from g_i mod s_α, and the symbol α was replaced with the $(r_i + 1)$-th symbol in the candidate rule list. Following this translation rules, We can obtain the phenotype "$1/x - x$" from the genotype "6214331513" according to pre-order walk traverse as shown in Fig. 2.

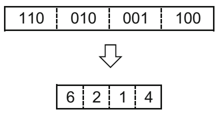


Fig. 1. Mapping from binary genotype to integer genotype.

Table 1. Translation rule in simple example

Rule	Candidate rule list	Rule no.
(A)	`<expr>` ::= `<expr><op><expr>` | `<var>`	(0) | (1)
(B)	`<op>` ::= `+` | `-` | `*` | `/`	(0) | (1) | (2) | (3)
(C)	`<var>` ::= `1` | `x`	(0) | (1)

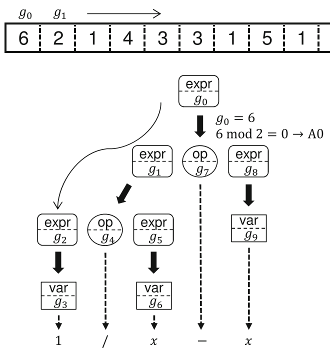


Fig. 2. Pre-order walk traverse of symbols from genotype.

2.2 Pointer Allocation Mechanism

Our proposal introduced a pointer allocation mechanism to map the tree structure from the chromosomes as shown in Fig. 3. The codons of `<expr>` are assigned into the tree structure following the linear-order walk traverse. The pointer p is firstly allocated to the header of chromosome e as follows:

$$p = e[0] \tag{1}$$

and

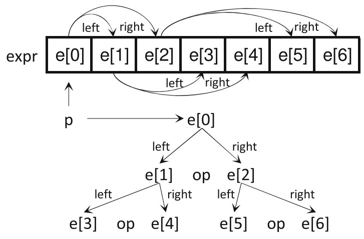
$$p \rightarrow left = e[1], \tag{2}$$

$$p \rightarrow right = e[2] \tag{3}$$

where $e[\ ]$ denotes the `<expr>` array. The p points the root of binary tree, and $p.left$ and $p.right$ points the left node and right node, respectively. As a tree is a self-referential data structure, the linear genotype structure can be allocated as a linked lists as follows:

$$p = p \rightarrow left, \tag{4}$$

$$p = p \rightarrow right. \tag{5}$$



expr
e[0] e[1] e[2] e[3] e[4] e[5] e[6]
left right left right left right
p
e[0]
left right
e[1] op e[2]
left right left right
e[3] op e[4] e[5] op e[6]

Fig. 3. Pointer allocation mechanism for tree structure.

2.3 Multiple Chromosome System

For different kinds of nonterminals, arrays $e[\ ], o[\ ]$ and $v[\ ]$ denote `<expr>` array, `<op>` array and `<var>` array, which were allocated to each sub-chromosome (Fig. 4). According to Table 1, our proposal separated the nonterminals into the recursive nonterminals and non-recursive nonterminals. `<expr>` is the recursive nonterminal, and `<op>` and `<var>` are the non-recursive nonterminals.

Due to recursive nonterminals were recursively replaced by themselves as shown in translation rule (A0), this type of rule is called recursive rule. When pointer p was allocated to any $e[i]$, allocation strategy for `<expr>::= <expr><op><expr>` could be defined as follows. The `<expr>` in the root is represented as

$$p = e[i], \tag{6}$$

and the two `<expr>` in the both sides are represented as

$$p \rightarrow left = e[2 * i + 1], \tag{7}$$

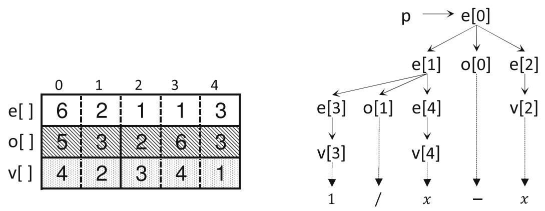
$$p \rightarrow right = e[2 * i + 2]. \tag{8}$$

Here, `<op>` in rule (A0) is non-recursive nonterminal, and is replaced by non-recursive nonterminals alone i.e. rule (A1). Due to non-recursive nonterminals would be translated into phenotype by terminals, this type of rule is called non-recursive rule. When pointer p was allocated to any $e[i]$, allocation strategy for `<op>` in `<expr>::= <expr><op><expr>` and `<expr>::= <var>` could be defined as follows:

$$p \rightarrow op = o[i] \tag{9}$$

and

$$p \rightarrow var = v[i]. \tag{10}$$



(a) Multi-chromosome (b) Pointer allocation

Fig. 4. Pointer allocation mechanism in multi-chromosome.

3 Experiment

3.1 Problems Setting

In the experiments, we used three kinds of symbolic regression and Santa Fe Ant Trail problem to compare the performance and effectiveness of GETRL with that of original GE and GE using multi-chromosome (GEMC). Parameter setting for

Table 2. Parameter settings

Parameter	Values
Maximum generations	500
Population size	200
Selection	Tournament selection
Tournament size	5
Number of elites	5
Crossover	Uniform crossover
Crossover rate (CR)	0.0, 0.1, 0.2, 0.3, 0.4, 0.5
Mutation rate (MR)	0.03, 0.05, 0.1

GE, GEMC and GETRL was shown in Table 2 and the fitness was based on root mean square error. In order to obtain the most approximative target function, we examined respective methods for 50 runs, and selected the best parameter for illustrating the results and discussions.

Symbolic Regression. The grammar used for symbolic regression is given as follow:

$$\begin{aligned} N &= \{\texttt{expr, op, var, num, char}\}, \\ T &= \{\texttt{+, -, *, /, \^{}, 1, x, y}\}, \\ S &= \{\texttt{expr}\}, \end{aligned}$$

and three kinds of symbolic regression are listed bellow.

Ex. 1:

$$f_1(x) = x + x^2 + x^3 + x^4.$$

Ex. 2:

$$f_2(x) = x^4 - 2x^3 + 3x^2 - 4x + 5.$$

Ex. 3:

$$f_3(x, y) = (x - y)^5.$$

Santa Fe Ant Trail Problem (Ex. 4). The grammar of Santa Fe Ant Trail problem [8] is given as follow:

$$\begin{aligned} N &= \{\texttt{code, op}\}, \\ T &= \{\texttt{if, else, food\_ahead, turn\_left,} \\ &\quad \texttt{turn\_right, move}\}, \\ S &= \{\texttt{code}\}, \end{aligned}$$

and the fitness is calculated by Eq. (11),

$$f = F - F_{max} \tag{11}$$

where $F_{max} = 89$ denotes all the pieces of food, and F denotes the obtained pieces of food.

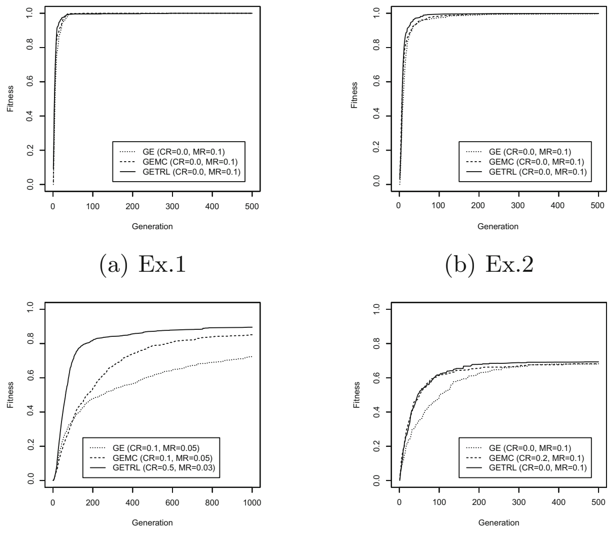
3.2 Results

Ex. 1. Ex.1 presented the standard quartic symbolic regression problem, which is widely used as the benchmark. However, either of these three methods can find the appropriate function to the target function, GETRL showed a better convergence than GE and GEMC around the 10th–50th generation (Fig. 5(a)).

Ex. 2. Ex.2 presented another quartic symbolic regression problem. Ex.2 used the same BNF syntax as Ex.1, but the expression is more complicated than Ex. 1. In Ex.2 (Fig. 5(b)), GETRL also showed a better convergence than GE and GEMC from the 10th generation. The convergence speed of GETRL to find the appropriate function to the target function around the 100th generation is also much faster than GE and GEMC, in which the appropriate function can be found to the target function around the 200th generation. Due to the proposed pointer allocation strategy, GETRL outperformed GE and GEMC in preventing the invalid individuals, especially at the beginning of search.

Ex. 3. Ex.3 presented a symbolic problem with two variables. However, GETRL showed a worse convergence than GE and GEMC at the beginning, GETRL became much superior than GE and GEMC from the 20th generation, and got the appropriate function around 480th generation (Fig. 5(c)). Accounting for its pointer allocation strategy, GETRAL allowed the offspring to remain the effective schemata and showed a dramatically performance than GE and GEMC.

Ex. 4. Figure 5(d) showed the comparison of GE, GEMC and GETRL in Santa Fe Ant Trail problem. GETRL outperformed GE around the 10th generation,



(a) Ex.1 (b) Ex.2

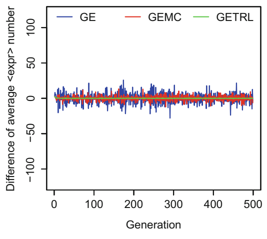
(c) Ex.3 (d) Ex.4

Fig. 5. Comparison of convergence.

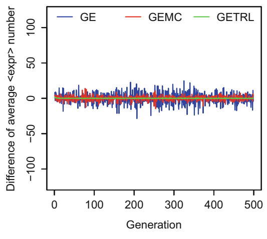
and outperformed GEMC around the 100th generation. The pointer allocation strategy in GETRL contributed to this problem into two folds. One is the effectiveness of remaining the schemata from parents. The other one is the utility of preventing to read invalid genes, which can lead it to producing more efficient program than GE and GEMC.

3.3 Discussion

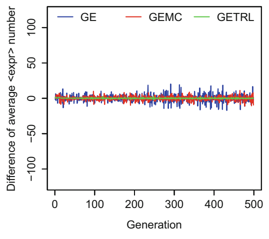
Symbolic Regression. Figure 6 showed the effectiveness of GETRL comparing with GE and GEMC. First, we subtracted average number of `<expr>` using in each generation from the previous generation. We estimated the difference of the average number between two generations, and the smaller change of this value indicated the more similarity of offspring inherited from the parents. The difference in each generation of Ex.1, Ex.2 and Ex.3 were represented in Figs. 6(a), (b) and (c), respectively. Second, we also investigated individuals to read the invalid genes resulting death. The number of dead individuals in each generation of Ex.1, Ex.2 and Ex.3 were represented in Fig. 6(d), (e) and (f). Due to the pointer allocation mechanism, GETRL was able to outperform GE and GEMC in inheriting better schemata from parents and preventing invalid individuals in offspring.



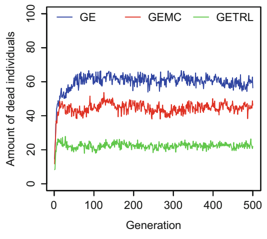
(a) Ex.1



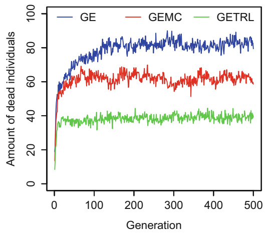
(b) Ex.2



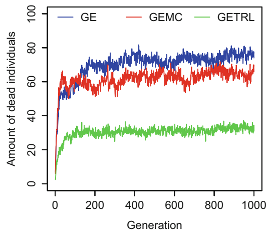
(c) Ex.3



(d) Ex.1



(e) Ex.2



(f) Ex.3

Fig. 6. Symbolic regression problem. (a), (b) and (c): difference of average <expr> number in Ex.1, Ex.2 and Ex.3; (d), (e) and (f): amount of dead individuals in Ex.1, Ex.2 and Ex.3.

Santa Fe Ant Trail Problem. As shown in Fig. 7, average amount of foods eaten was calculated for 50 runs. The maximum values and minimum values in these 50 runs were extracted, which denoted foods eaten by using the best program

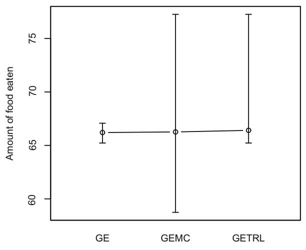


Fig. 7. Amount of foods eaten in Santa Fe Ant Trail problem.

and the worst program generated from GE, GEMC, and GETRL, respectively. As GETRL obtained best results in either of the maximum, minimum and average amount of foods eaten, the pointer allocation mechanism in GETRL could lead to the production of a more superior program than GE and GEMC.

4 Conclusion

This paper described an improved algorithm of grammatical evolution using tree representation learning (GETRL). GETRL presented a novel approach to the GE, which adopted pointer allocation mechanism and multiple chromosomes system. The pointer allocation mechanism was helpful for recording the position of individual genes in the chromosome and mapping the tree structure from genotype instead of list or array structure. The multiple chromosomes system was used for enhancing the effectiveness of GETRL, and better preserving the partial structures of solutions when genetic operators were applied. We compared the performance of GETRL with that of the original GE and GE using multi-chromosomes for the symbolic regression problems and Santa Fe Ant Trail problem. Due to GETRL perfectly resolved the poor search properties, which are resulted from the agnostic link between the linear representation in GE and the derivation tree of phenotype, our method showed faster convergence in finding the appropriate solutions.

For future work, there remained two limitations to be addressed. First, however the point allocation mechanism can prevent reading of invalid genes. As the tree structure is a fixed binary tree, it is easy to cause the pointer index out of range. Next, the recursive rules of BNF syntax design must contain two recursive nonterminals such as `<expr><op><exper>` or `<code><code>`, for fitting to binary tree structure. Therefore, we plan to propose a self-adaptive pointers and memory allocation strategy with two operations *malloc* and *free*. According to any grammar, the memory can be dynamically allocated by *malloc* to generate a dynamic tree structure. After genetic operations, the unused memory can be deallocated by *free* to prevent from out of range.

References

1. Bäck, T., Schwefel, H.P.: An overview of evolutionary algorithms for parameter optimization. Evol. Comput. **1**(1), 1–23 (1993)

2. Brabazon, A., O'Neill, M.: Biologically Inspired Algorithms for Financial Modelling. Natural Computing Series. Springer-Verlag New York Inc., Secaucus (2006). doi:10.1007/3-540-31307-9
3. Byrne, J., O'Neill, M., McDermott, J., Brabazon, A.: An analysis of the behaviour of mutation in grammatical evolution. In: Esparcia-Alcázar, A.I., Ekárt, A., Silva, S., Dignum, S., Uyar, A.Ş. (eds.) EuroGP 2010. LNCS, vol. 6021, pp. 14–25. Springer, Heidelberg (2010). doi:10.1007/978-3-642-12148-7_2
4. Holland, J.H.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. MIT Press, Cambridge (1992)
5. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
6. Lourenço, N., Pereira, F.B., Costa, E.: Unveiling the properties of structured grammatical evolution. Genet. Program Evolvable Mach. **17**(3), 251–289 (2016)
7. Murphy, E., O'Neill, M., Galvan-Lopez, E., Brabazon, A.: Tree-adjunct grammatical evolution. In: IEEE Congress on Evolutionary Computation, pp. 1–8 (2010)
8. O'Neill, M., Ryan, C.: Grammatical evolution. IEEE Trans. Evol. Comput. **5**(4), 349–358 (2001)
9. O'Neill, M., Brabazon, A., Nicolau, M., Garraghy, S.M., Keenan, P.: πGrammatical evolution. In: Deb, K. (ed.) GECCO 2004. LNCS, vol. 3103, pp. 617–629. Springer, Heidelberg (2004). doi:10.1007/978-3-540-24855-2_70
10. O'neill, M., Ryan, C., Keijzer, M., Cattolico, M.: Crossover in grammatical evolution. Genet. Program Evolvable Mach. **4**(1), 67–93 (2003)
11. Ryan, C., Azad, A., Sheahan, A., O'Neill, M.: No coercion and no prohibition, a position independent encoding scheme for evolutionary algorithms – the Chorus system. In: Foster, J.A., Lutton, E., Miller, J., Ryan, C., Tettamanzi, A. (eds.) EuroGP 2002. LNCS, vol. 2278, pp. 131–141. Springer, Heidelberg (2002). doi:10.1007/3-540-45984-7_13
12. Ryan, C., Collins, J.J., Neill, M.O.: Grammatical evolution: evolving programs for an arbitrary language. In: Banzhaf, W., Poli, R., Schoenauer, M., Fogarty, T.C. (eds.) EuroGP 1998. LNCS, vol. 1391, pp. 83–96. Springer, Heidelberg (1998). doi:10.1007/BFb0055930
13. Ryan, C., O'Neill, M., Collins, J.: Grammatical evolution: solving trigonometric identities. In: Proceedings of Mendel 1998: 4th International Conference on Genetic Algorithms, Optimization Problems, Fuzzy Logic, Neural Networks and Rough Sets, pp. 111–119 (1998)
14. Schwefel, H.P.P.: Evolution and Optimum Seeking: The Sixth Generation. Wiley, New York (1993)
15. Shaker, N., Nicolau, M., Yannakakis, G.N., Togelius, J., O'Neill, M.: Evolving levels for super mario bros using grammatical evolution. In: 2012 IEEE Conference on Computational Intelligence and Games (CIG), pp. 304–311 (2012)
16. Thorhauer, A., Rothlauf, F.: On the locality of standard search operators in grammatical evolution. In: Bartz-Beielstein, T., Branke, J., Filipič, B., Smith, J. (eds.) PPSN 2014. LNCS, vol. 8672, pp. 465–475. Springer, Cham (2014). doi:10.1007/978-3-319-10762-2_46
17. Whigham, P.A., Dick, G., Maclaurin, J., Owen, C.A.: Examining the "best of both worlds" of grammatical evolution. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, GECCO 2015, pp. 1111–1118. ACM, New York (2015)