Exploring Random Permutations Effects on the Mapping Process for Grammatical Evolution

Submitted: 20th December 2019; accepted: 30th March 2020

Blanca Verónica Zúñiga, Juan Martín Carpio, Marco Aurelio Sotelo-Figueroa, Andrés Espinal, Omar Jair Purata-Sifuentes, Manuel Ornelas, Jorge Alberto Soria-Alcaraz, Alfonso Rojas

DOI: 10.14313/JAMRIS/1-2020/8

Abstract: *Grammatical Evolution (GE) is a form of Genetic Programming (GP) based on Context-Free Grammar (CF Grammar). Due to the use of grammars, GE is capable of creating syntactically correct solutions. GE uses a genotype encoding and is necessary to apply a Mapping Process (MP) to obtain the phenotype representation. There exist some well-known MPs in the state-of-art like Breadth-First (BF), Depth-First (DF), among others. These MPs select the codons from the genotype in a sequential manner to do the mapping. The present work proposes a variation in the selection order for genotype's codons; to achieve that, it is applied a random permutation for the genotype's codons order-taking in the mapping. The proposal's results were compared using a statistical test with the results obtained by the traditional BF and DF using the Symbolic Regression Problem (SRP) as a benchmark.*

Keywords: *Grammatical evolution, mapping process, symbolic regression*

1. Introduction

Genetic Programming (GP) is an Automatic Programming (AP) technique proposed by Koza [1]. It aims the automatic construction of solutions to different types of problems. One way to obtain syntactically correct solutions is by using grammars that restrict the search space [2, 3]. Grammars provide a mechanism that can be used to describe complex structures and define what can be done [4]. Variants which are grammar-based are the second most commonly used variations of GP [5].

Grammatical Evolution (GE) [6] is a GP based form that uses an integer string and a grammar in a genotype-phenotype mapping to obtain syntactically correct and feasible sentences [7]. Unlike GP, GE performs the evolutionary process in the linear genotype rather than the solution [6]. The Mapping Process (MP) is GE's component that allows generating solutions (phenotypes) that are guaranteed to be syntactically correct from an integer string (genotype) [8]. This MP can be seen as an abstraction of the DNA, is the conversion of a chromosome (genotype) to a solution (phenotype) [9].

The original MP used in GE was the Depth-First (DF) MP [6]. DF creates the phenotype by taking in linear order one codon value to select one grammar's production rule applying an equation. Breadth-First (BF) MP [10] was proposed later. The only difference between these two MPs is the order in which the expansion is carried out.

DF and BF are considered the classic MPs, and both use a Backus Naur Form Grammar (BNF-Grammar); since then, many other approaches have been proposed, like the πGrammatical Evolution (πGE) [11], it employs two codons rather than just one: the first codon is used to select the non-terminal to expand (main difference with the classic MPs), and the second codon is used in the same way as the last MPs to select a production rule from the grammar; the Tree-Adjunct Grammatical Evolution (TAGE) [12] uses a tree-adjunct grammar instead of a BNF-Grammar to create the phenotype; the Univariate Model-Based Grammatical Evolution (UMBGE) [13] uses probabilistic context-free grammars and replaces the original genetic operators with the sampling from the distribution of the best solutions; Structured Grammatical Evolution (SGE) [14] uses a different genotypic representation for GE, where each gene is explicitly linked to a non-terminal of the grammar with the purpose of increasing locality; among others. There exist some studies about the performance of these different MPs [15, 16, 10] used to solve various types of problems, such as the Symbolic Regression Problem (SRP), The Santa Fe Ant Trail, and the Even-five Parity Problem. In the classic MPs, the mapping is performed by taking each integer value of the genotype (called codon), and an equation to choose the corresponding production rule. It is created a derivative tree from this process, and the solution taken from it. The order-taking for the genotype's codons in the classic MPs DF and BF is sequential.

In this paper, we propose a modification in the order-taking of the genotype's codons for the MPs DF and BF. The obtained results are compared with these traditional MPs applied to the SRP using a statistical test. The paper is structured as follows: Section 2 gives a brief introduction to GE and its components, in Section 3 is presented the proposed approach, the selected setup that was used in the experiments is explained in Section 4; Section 5 presents the obtained results and the statistical analysis, and, finally the conclusions and future work are discussed in Section 6.

2. Grammatical Evolution

GE is a variant of GP that takes inspiration from the biological evolutionary process (a comparative is shown in Figure 1), in GE the DNA is represented as an integer string; to replicate the process, GE uses an MP and a type of grammar to produce the phenotypic solution [6].

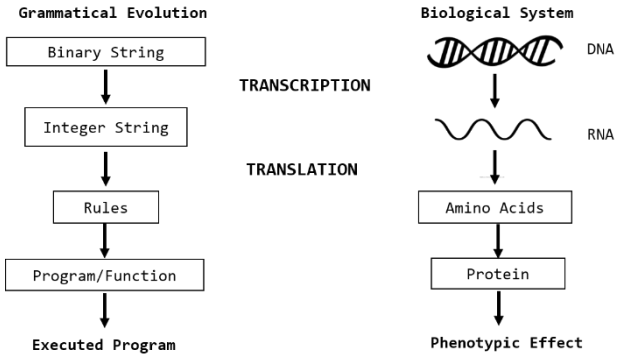

Fig. 1. Comparison between the GE approach and a biological genetic system [6]

Traditionally, a BNF-Grammar is used to create syntactically correct solutions [6]. The BNF-Grammar provides the necessary rules to produce the phenotype according to the specific problem that is trying to solve [17]. GE has three main components [18], but for the aim of this work, it has been added the MP as the fourth component. Figure 2 presents the used methodology for GE, it needs four main components: the Problem Instance, a BNF-Grammar, the Search Engine, and the MP.

Due to the modular nature of GE, each one of the components used in GE can be switched [19], from the Problem Instance to the MP.

GE produces as an output a phenotype, which represents the solution found, later, this solution is evaluated with the objective function, and the process continues until the stopping condition is met. The Algorithm 1 shows GE's algorithm.

Algorithm 1 Grammatical Evolution Algorithm

Require: The number of elements of the initial population as *pop_size*
Population = new_population(*pop_size*)
Solution_found = false
Initialize (Population)
while *termination condition not satisfied* ≥0 **do**
  PerformMappingProcess(Population)
  Evaluate(Population)
  if *Solution_found* **then**
    Return Best Solution
    end
  end if
  if Solution_found == false **then**
    PerformGeneticOperators(Population)
  end if
end while
Return the best solution found

2.1. Problem Instance

The Problem Instance refers to the type of issue that is trying to get solved. For example, problems like the Bin Packing Problem (BPP) [18], Even-5-parity [15, 10], the Santa Fe Ant Trail (SFAT) problem [15, 10], Data Classification [20], the design of the topology of Artificial Neural Networks [21, 22], the Flexible Job Shop Scheduling Problem [23], and the SRP [24, 25, 26, 15] have been tried to get solved with GE.

The Problem used in the present work is SRP, explained in Section 2.1.1.

Symbolic Regression Problem. The SRP [1] is one of the most requested problem domains in the GP community [5]. Techniques like GP [27, 28] and GE [24] have been used to solve such task.

SRP intends to find a mathematical expression that represents (with the minimal error) a given set of data that takes as a base the rules of accuracy, simplicity, and generalization [28]. The obtained mathematical expression can be seen as a function that takes the values of the variables as an input and returns an output [1].

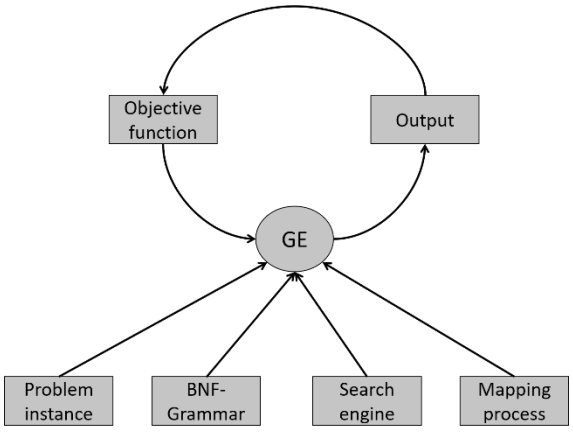

Fig. 2. Used GE's methodology based on [18]

2.2. Backus Naur Form Grammar

BNF-Grammar is a type of grammar employed in GE. The next tuple form this type of grammar [2]:

$$G = \{NT, T, R, S\}$$

where:
NT is the set of non-terminal symbols.
T is the set of terminal symbols.
R corresponds to the production rules.
S corresponds to the start symbol, S ∈ NT.

Initially, a BNF-Grammar must be defined. This grammar specifies the structure that the possible solutions produced by GE must have.

The BNF-Grammar consist of two types of symbols, the non-terminal (NT) symbols, and the terminal (T) symbols. The first type can be expanded into NT or T symbols (according to the production rules of the grammar); the second type corresponds to the set that contains the symbols that are allowed to appear in the final expression. The last one, the start symbol S indicates where is the starting point in the grammar. As an example, in Grammar 1 the NT is the set described by NT = {<e>, <v>, <o>}, and the set of T is described by T = {X, Y, -, +}.

Each NT has its corresponding production rules (separated by the symbol "⊨"), and each production rule is separated by the symbol "|".

```
NT = {<start>, <e>, <v>, <o>}
T = {x, y, +, -}
S = {<start>}
P = set of production rules

<start>⊨      <e>           (0)

<e>     ⊨     <e> <o> <e>   (0)
        |     <v>           (1)
<v>     ⊨     x             (0)
        |     y             (1)
<o>     ⊨     +             (0)
        |     -             (1)
```

Grammar 1. Example of a simple BNF-Grammar

2.3. Search Engine

The main intention of the Search Engine (SE) is to evolve the candidates throw a search algorithm to find the best one [6]. To achieve that, the SE evaluates each candidate with the objective function, this evaluation is called fitness and represents the performance of an individual to solve a determinate problem [29, 30, 6].

Genetic Algorithm. In this work is used the Genetic Algorithm (GA) as SE. The reason to use the GA is that it represents the canonical search algorithm used in GE's initials [6]. GA is a metaheuristic inspired by the evolutionary process proposed in its initials by Darwin [31]; Holland later proposed the algorithm for the GA [32]. Algorithm [2] shows the process of GA.

Algorithm 2 Genetic Algorithm

```
Require: The number of elements of the initial population as pop_size
Population = new_population(pop_size)
Solution_found = false
Initialize (Population)
while termination condition not satisfied ≥ 0 do
    PerformGeneticOperators(Population)
    Evaluate(Population)
    if Solution_found then
        Return Best Solution
    end
    end if
end while
Return the best solution found
```

2.4. Mapping Process

The MP is the procedure of creating a derivative tree with the help of a grammar. GE uses the Equation 1, a grammar and the genotype (an integer string) to transform the genotype into a phenotype. In Figure 3 is shown an example of this process.

$$ProdRule = CodonVal \ \% \ NumOfProdRule \ for \ the \ NT. \quad (1)$$

To exemplify each MP, it is used the example Grammar 1, and the following genotype:

Genotype = 2,12,7,9,3,15,23,1,11,4,6,13,2,7,8,3,35,19,2,6

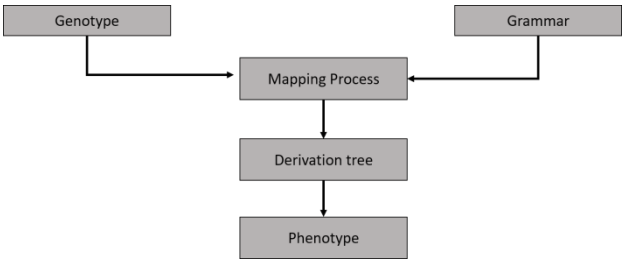

Fig. 3. Used GE's methodology based on [18]

Depth-First Mapping Process. The DF MP [6] starts from the start symbol and makes the corresponding expansion by taking the left-most NT in the derivative tree. Equation 1 is used to choose the appropriate production rule, by substituting the corresponding codon value of the genotype and the number of production rules of the current NT. This process is presented in Figure 4 (the numbers out of the parenthesis indicate the expansion order in the derivative tree). In the example we start with the NT <e>; using the Eq. 1 we substitute the corresponding values, taking the first codon value: 2, and the number of production rules for <e>: 2, the result is 0, which indicates that the next production rule is the one that corresponds to the position zero. And now, there are three new available NTs in the list: <e>, <o> and <e>; the next NT to expand is <e>. We repeat this same process until there no more NTs remain in the derivative tree. The phenotype is obtained by traversing the end nodes of the expansion tree. The obtained phenotype for this example is "Y-Y-X". DF is considered the classic MP for GE [6]. The corresponding algorithm for this MP is shown in Algorithm 3.

Algorithm 3 Depth-First Mapping Process Algorithm.

```
Require: the BNF-Grammar as grammar, and the integer string as genotype.
listNT{List to store NTs seen}
Add start symbol from grammar to listNT
wraps = 0
while listNT is not empty do
    if reached end of chromosome then
        wraps++
        if wraps> max wraps allowed then
            return false
        end if
        reset chromosome iterator
    end if
    CurrentNT = get head of listNT
    CurrentCodon = get next codon value
    newProduction = currentCodon % number of productions for currentNT
    set currentNTs children = newProduction
    add newProduction to head of listNT {Only adds NTs}
end while
Generate phenotype by traversing the leaf nodes of the derivation tree
return true
```

Breadth-First Mapping Process. The second MP used in this work is the BF MP [10]. This MP distinguishes from the DF only by the order in which the expansion is executed. It uses the same equation to choose the corresponding production rule (Eq. 1) but makes the expansion level by level in a left to right order.

In the same way as the last process, the mapping initiates with the start symbol <e> (specified in the grammar). Applying the module rule the corresponding expansion is <e><o><e>. The expansion continues

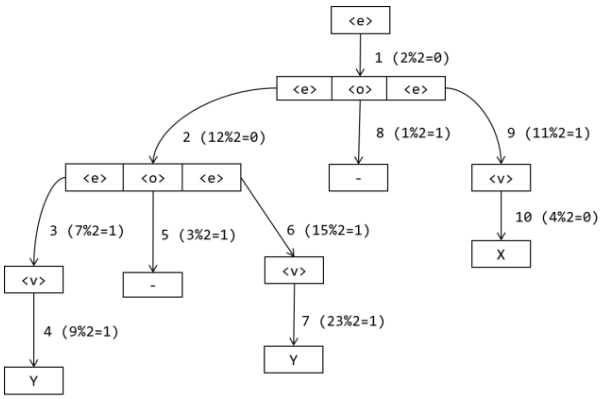

Fig. 4. Example of Depth-First Mapping Process [6]

by taking every NT in a level of the tree from left to right until no more NTs remain. In this case, the retrieved phenotype is "Y-X-Y". The example of this process is shown in Figure 5, and its corresponding algorithm in Algorithm 4.

Algorithm 4 Breadth-First Mapping Process Algorithm

Require: the BNF-Grammar as *grammar*, and the integer string as *genotype*.
1: *listNT*{List to store NTs seen}
2: Add start symbol from grammar to *listNT*
3: wraps = 0
4: **while** *listNT* is not empty **do**
5: if reached end of chromosome **then**
6: *wraps++*
7: if *wraps*> max wraps allowed **then**
8: **return false**
9: end if
10: reset chromosome iterator
11: end if
12: *CurrentNT* = get head of *listNT*
13: *CurrentCodon* = get next codon value
14: *newProduction* = *currentCodon % number of productions for currentNT*
15: set *currentNTs* children = *newProduction*
16: add *newProduction* to tail of *listNT* {Only adds NTs}
17: **end while**
18: Generate phenotype by traversing the leaf nodes of the derivation tree
19: **return true**

3. Proposed Approach

In the classical MPs BF and DF, the codons are taken sequentially. It means that each codon value is used one by one in order of appearance as is shown in the example Figure 6. The figure shows the genotype (represented as an integer string in the first row), its corresponding sequential order-taking for the codons (second row), the BNF-Grammar employed, and the correspondent derivative process. In this last, the list of NTs is placed at the left side, and on the right side is indicated the order in which the codons are taken. Before the "→" symbol, the corresponding substituted values are shown for the Equation 1.

In the proposed approach, we change the order-taking for the codons in the MPs BF and DF. To set this order-taking is employed a random permutation.

Figure 7 shows an example of the proposal used in the DF MP. The figure shows the genotype (represented as an integer string in the second row), its corresponding order-taking for the codons (third row), the original index for the genotype (first row), the BNF-Grammar employed, and the derivative process.

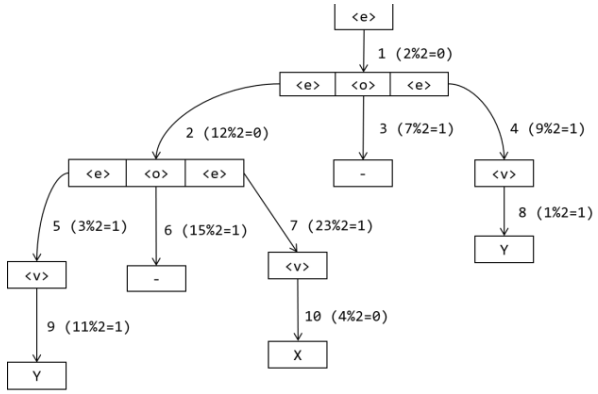

Fig. 5. Example of Breadth-First Mapping Process

Algorithm 5 Proposed Algorithm for the Depth-First Mapping Process.

Require: the BNF-Grammar as *grammar*, and the integer string as *genotype*.
1: *listNT*{List to store NTs seen}
2: Add start symbol from grammar to *listNT*
3: **Create the random integer string for the permutations as *positions***
4: **while** *listNT* is not empty **do**
5: if reached end of chromosome **then**
6: end Mapping Process
7: **return phenotype**
8: **else**
9: *CurrentNT* = get head of *listNT*
10: *CurrentCodon* = get next codon value from *genotype* in the order of *positions*
11: *newProduction* = *currentCodon % number of productions for currentNT*
12: set *currentNTs* children = *newProduction*
13: add *newProduction* to head of *listNT* {Only adds NTs}
14: **end if**
15: **end while**
16: Generate *phenotype* by traversing the leaf nodes of the derivation tree
17: **return *phenotype***

Algorithm 6 Proposed Algorithm for Breadth-First Mapping Process

Require: the BNF-Grammar as *grammar*, and the integer string as *genotype*.
1: *listNT*{List to store NTs seen}
2: Add start symbol from grammar to *listNT*
3: **Create the random integer string for the permutations as *positions***
4: **while** *listNT* is not empty **do**
5: if reached end of chromosome **then**
6: End Mapping Process
7: **return phenotype**
8: **else**
9: *CurrentNT* = get head of *listNT*
10: *CurrentCodon* = get next codon value from *genotype* in the order of *positions*
11: *newProduction* = *currentCodon % number of productions for currentNT*
12: set *currentNTs* children = *newProduction*
13: add *newProduction* to tail of *listNT* {Only adds NTs}
14: **end if**
15: **end while**
16: Generate *phenotype* by traversing the leaf nodes of the derivation tree
17: **return *phenotype***

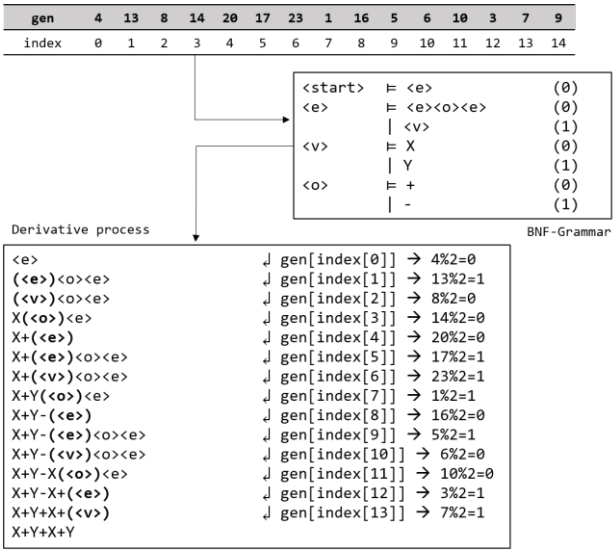

Fig. 6. Example of the transformation genotype-to-phenotype using the classic MP DF

In this last, the list of NTs is placed at the left side, and on the right side is indicated the order in which the codons are taken. Note that it is used the permutation (perm) to choose the index of the gen. After the "→" symbol, the corresponding substituted values are shown for the Equation 1.

4. Experimental Analysis

4.1. Benchmark Functions

Experiments were performed to evaluate the performance of the proposal using a set of ten functions of the SRP. Table 1 shows the ten functions used in the experimental analysis. These functions were taken from [24, 33].

Tab. 1. Symbolic Regression functions used as instance set [24, 33]

Function	Fit Cases
$F_1 = x^3 + x^2 + x$	20 random points x ∈ [-1,1]
$F_2 = x^4 + x^3 + x^2 + x$	
$F_3 = x^5 + x^4 + x^3 + x^2 + x$	
$F_4 = x^6 + x^5 + x^4 + x^3 + x^2 + x$	
$F_5 = \sin(x^2)\cos(x) - 1$	
$F_6 = \sin(x)\sin(x + x^2)$	
$F_7 = \log(x+1) + \log(x^2+1)$	20 random points x ∈ [0,2]
$F_8 = \sqrt{x}$	20 random points x ∈ [0,4]
$F_9 = \sin(x) + \sin(y^2)$	200 random points ∈ [-1,1], x ∈ [-1,1], y ∈ [-1,1]
$F_{10} = 2\sin(x)\cos(y)$	

4.2. Parameter Setup

We employed a GA as a Search Engine for the GE in the experiments to make the evolutionary process of the genotypes. The used parameters were set empirically. Table 2 shows the corresponding parameter values for the GA.

Grammar 2 shows the grammar used in the experiments. This grammar was taken from [24].

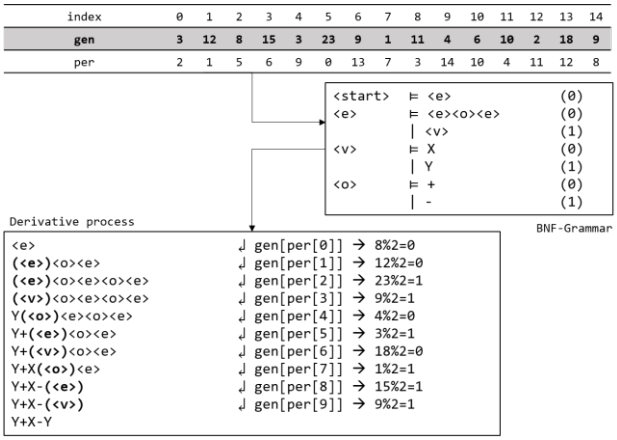

Fig. 7. Example of the transformation genotype-to-phenotype using the proposal with the DF MP

Tab. 2. Parameter settings used in the GA

Parameter	Value
Population size	300 individuals
Initial genotype length	100 codons (random init)
Stopping condition	25,000 function calls
Selection Method	Binary tournament
Crossover Operator	2 points
Mutation Operator	Flip bit
Replacement Strategy	Generational with elitism (best individual)

Tab. 3. Medians and variances for the best fitness in classical MPs (DF and BF) and these MPs using the proposal applied to the SRP

Function	Classic DF		Classic BF		DF Proposal		BF Proposal	
	Median	Var	Median	Var	Median	Var	Median	Var
F1	0.0056	0.0001	0.0045	0.0001	0.0066	1.85E-05	0.0056	9.50E-06
F2	0.0845	0.0229	0.0101	0.0014	0.0266	0.0002	0.0239	0.0003
F3	0.0260	0.0014	0.0116	0.0003	0.0125	7.70E-05	0.0159	2.62E-05
F4	0.1452	0.0714	0.0345	0.0062	0.0211	0.0007	0.0211	0.0003
F5	0.0032	0.0001	0.0034	5.72E-05	0.0025	1.81E-06	0.0024	1.58E-06
F6	0.0065	0.0024	0.0140	0.0008	0.0132	7.02E-05	0.0220	0.0001
F7	0.0013	7.79E-05	0.0029	0.0017	0.0018	2.63E-07	0.0016	2.94E-07
F8	0.0177	0.0025	0.0122	0.0008	0.0039	2.14E-05	0.0039	1.46E-05
F9	0.0028	0.0011	0.0081	0.0012	0.0023	1.56E-06	0.0023	8.94E-07
F10	0.0030	0.0012	0.0023	0.0015	0.0030	1.67E-07	0.0030	8.62E-07

```
NT = {<start>, <expr>, <var>, <pre-op>, <op>}
T = {x, y, 1.0, sin, cos, exp, log, +, -, /, *}
S = {<start>}
P = set of production rules

<start>  ⊨      <expr>                    (0)
<expr>   ⊨      (<expr><op><expr>)        (0)
         |      <pre-op>(<expr>)          (1)
         |      <var>                     (2)
<var>    ⊨      x                        (0)
         |      y                        (1)
         |      1.0                      (2)
<pre-op> ⊨      sin                      (0)
         |      cos                      (1)
         |      exp                      (2)
         |      log                      (3)
<op>     ⊨      +                        (0)
         |      -                        (1)
         |      /                        (2)
         |      *                        (3)
```

Grammar 2. Grammar used for the SRP

The Mean Root Squared Error (MRSE) given by Equation 2 was used as objective function to evaluate the candidate expressions obtained by the GE.

$$MRSE = \sqrt{\frac{\sum_{i=1}^{N}(y_i - F(x_i))^2}{N}} \quad (2)$$

where:
N is the number of data points.
y_i is the real value.
$F(x_i)$ corresponds to the obtained value.

5. Results and Statistical Analysis

33 individual experiments were performed to evaluate the performance of the proposal. Table 3 shows the obtained results (the median and variance of the best fitness values achieved) using the classical MPs and the MPs with the proposal to solve each function in Table 1.

A Friedman non-parametrical test was used to know if there exists a significant difference between the proposal and the classical MPs. The obtained p-value for the medians of the best fitness was 0.5163.

The same test was performed for the variances of the best fitness. The obtained p-value was 1.87E-05. Table 4 shows the average ranking obtained with the variances.

Tab. 4. Average rankings of the MPs

Mapping Process	Ranking
DF Proposal	1.4
BF Proposal	1.6
Classic DF	3.3
Classic BF	3.7

6. Conclusion

A new approach for the order-taking of codons in the MPs Depth-First and Breadth-First applied to the SRP was proposed. The obtained results were compared with the well-known MPs Depth-First and Breadth-First using the Friedman non-parametrical test.

Derived from the obtained results with the Friedman test we could conclude that there is no evidence to differentiate between the performance (regarding with the median) of the standard MPs, and this same MPs using the proposal.

However, there is statistical evidence to discern between the performance of the MPs concerning the variance. The results indicate that the proposal provides the algorithms of BF and DF MPs with higher consistency.

As future work, it is proposed to search for a methodology that helps to find the best permutation for the order-taking of the codons in GE's MPs.

ACKNOWLEDGEMENTS

The authors want to thank National Council for Science and Technology of Mexico (CONACyT) through the scholarship for postgraduate studies: 703582 (B. Zuñiga) and the Research Grant CÁTEDRAS-2598 (A. Rojas), the León Institute of Technology (ITL), and the Guanajuato University for the support provided for this research.

AUTHORS

Blanca Verónica Zúñiga – Postgraduate Studies and Research Division, León Institute of Technology, León, México, e-mail: m18240006@itleon.edu.mx.

Juan Martín Carpio – Postgraduate Studies and Research Division, León Institute of Technology, León, México, e-mail: juanmartin.carpio@itleon.edu.mx.

Marco Aurelio Sotelo-Figueroa* – Organizational Studies Department, DCEA-University of Guanajuato, Guanajuato, México, e-mail: masotelo@ugto.mx.

Andrés Espinal – Organizational Studies Department, DCEA-University of Guanajuato, Guanajuato, México, e-mail: aespinal@ugto.mx.

Omar Jair Purata-Sifuentes – Organizational Studies Department, DCEA-University of Guanajuato, Guanajuato, México, e-mail: opurata@yahoo.com.

Manuel Ornelas – Postgraduate Studies and Research Division, León Institute of Technology, León, México, e-mail: mornelas67@yahoo.com.mx.

Jorge Alberto Soria-Alcaraz* – Organizational Studies Department, DCEA-University of Guanajuato, Guanajuato, México, e-mail: jorge.soria@ugto.mx.

Alfonso Rojas – Postgraduate Studies and Research Division, León Institute of Technology, León, México, e-mail: alfonso.rojas@gmail.com.

* Corresponding author

REFERENCES

[1] J. R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, A Bradford Book, 1992.

[2] E. A. P. Hemberg, "An Exploration of Grammars in Grammatical Evolution". PhDthesis, University College Dublin, 2010.

[3] P. A. Whigham et al., "Grammatically-based Genetic Programming", Proceedings of the workshop on genetic programming: from theory to real-world applications, vol. 16, 33–41, 1995.

[4] M. O'Neill and C. Ryan, Grammatical Evolution, Springer US, 2003, DOI: 10.1007/978-1-4615-0447-4.

[5] D. R. White, J. McDermott, M. Castelli, L. Manzoni, B. W. Goldman, G. Kronberger, W. Jaśkowski, U.-M. O'Reilly and S. Luke, "Better GP benchmarks: community survey results and proposals", Genetic Programming and Evolvable Machines, vol. 14, no. 1, 2013, 3–29, DOI: 10.1007/s10710-012-9177-2.

[6] M. O'Neill and C. Ryan, "Grammatical evolution", IEEE Transactions on Evolutionary Computation, vol. 5, no. 4, 2001, 349–358, DOI: 10.1109/4235.942529.

[7] C. Ryan, M. O'Neill and J. Collins, "Introduction to 20 Years of Grammatical Evolution". In: C. Ryan, M. O'Neill and J. Collins (eds.), Handbook of Grammatical Evolution, 2018, 1–21, DOI: 10.1007/978-3-319-78717-6_1.

[8] M. O'Neill and A. Brabazon, "Grammatical Differential Evolution". In: H. R. Arabnia (eds.), Proceedings of the 2006 International Conference on Artificial Intelligence, ICAI 2006, Las Vegas, Nevada, USA, June 26-29, 2006, Volume 1, 2006, 231–236.

[9] D. Fagan and E. Murphy, "Mapping in Grammatical Evolution". In: C. Ryan, M. O'Neill and J. Collins (eds.), Handbook of Grammatical Evolution, 2018, 79–108, DOI: 10.1007/978-3-319-78717-6_4.

[10] D. Fagan and M. O'Neill, "Analyzing the Genotype-Phenotype Map in Grammatical Evolution", PhD thesis, University College Dublin, Oct. 2013.

[11] M. O'Neill, A. Brabazon, M. Nicolau, S. M. Garraghy and P. Keenan, "πGrammatical Evolution". In: K. Deb (eds.), Genetic and Evolutionary Computation – GECCO 2004, vol. 3103, 2004, 617–629, DOI: 10.1007/978-3-540-24855-2_70.

[12] E. Murphy, M. O'Neill, E. Galvan-Lopez and A. Brabazon, "Tree-adjunct grammatical evolution". In: IEEE Congress on Evolutionary Computation, 2010, 1–8, DOI: 10.1109/CEC.2010.5586497.

[13] H.-T. Kim and C. W. Ahn, "UMBGE: Univariate Model Based Grammatical Evolution", Journal of Computational and Theoretical Nanoscience, vol. 13, no. 7, 2016, 4104–4110, DOI: 10.1166/jctn.2016.5257.

[14] N. Lourenço, F. B. Pereira and E. Costa, "SGE: A Structured Representation for Grammatical Evolution". In: S. Bonnevay, P. Legrand, N. Monmarché, E. Lutton and M. Schoenauer (eds.), Artificial Evolution, vol. 9554, 2016, 136–148, DOI: 10.1007/978-3-319-31471-6_11.

[15] D. Fagan, M. O'Neill, E. Galván-López, A. Brabazon and S. McGarraghy, "An Analysis of Genotype-Phenotype Maps in Grammatical Evolution". In: A. I. Esparcia-Alcázar, A. Ekárt, S. Silva, S. Dignum and A. Ş. Uyar (eds.), Genetic Programming, vol. 6021, 2010, 62–73, DOI: 10.1007/978-3-642-12148-7_6.

[16] D. Fagan, "Genotype-phenotype Mapping in Dynamic Environments with Grammatical Evolution". In: Proceedings of the 13th annual conference companion on Genetic and evolutionary computation - GECCO '11, 2011, DOI: 10.1145/2001858.2002091.

[17] J. Hugosson, E. Hemberg, A. Brabazon and M. O'Neill, "Genotype representations in grammatical evolution", Applied Soft Computing, vol. 10, no. 1, 2010, 36–43, DOI: 10.1016/j.asoc.2009.05.003.

[18] M. A. Sotelo-Figueroa, H. J. Puga-Soberanes, J. M. Carpio, H. J. Fraire-Huacuja, L. Cruz-Reyes and J. A. Soria-Alcaraz, "Improving the Bin Packing Heuristic through Grammatical Evolution Based on Swarm Intelligence", Mathematical Problems in Engineering, 2014, 01–12, DOI: 10.1155/2014/545191.

[19] C. Ryan, M. O'Neill and J. Collins, Handbook of Grammatical Evolution, Springer International Publishing, 2018, DOI: 10.1007/978-3-319-78717-6.

[20] T. Chareka and N. Pillay, "A study of fitness functions for Data Classification using Grammatical Evolution". In: 2016 Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference (PRASA-RobMech), 2016, 1–4, DOI: 10.1109/RoboMech.2016.7813165.

[21] O. Quiroz-Ramírez, A. Espinal, M. Ornelas-Rodríguez, A. Rojas-Domínguez, D. Sánchez, H. Puga-Soberanes, M. Carpio, L. E. M. Espinoza and J. Ortíz-López, "Partially-Connected Artificial Neural Networks Developed by Grammatical Evolution for Pattern Recognition Problems". In: O. Castillo, P. Melin and J. Kacprzyk (eds.), Fuzzy Logic Augmentation of Neural and Optimization Algorithms: Theoretical Aspects and Real Applications, vol. 749, 2018, 99–112, DOI: 10.1007/978-3-319-71008-2_9.

[22] F. Ahmadizar, K. Soltanian, F. Akhlaghian Tab and I. Tsoulos, "Artificial neural network development by means of a novel combination of grammatical evolution and genetic algorithm", Engineering Applications of Artificial Intelligence, vol. 39, 2015, 1–13, DOI: 10.1016/j.engappai.2014.11.003.

[23] X. Li and L. Gao, "An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem", International Journal of Production Economics, vol. 174, 2016, 93–110, DOI: 10.1016/j.ijpe.2016.01.016.

[24] M. A. Sotelo-Figueroa, A. Hernández-Aguirre, A. Espinal, J. A. Soria-Alcaraz and J. Ortiz-López, "Symbolic Regression by Means of Grammatical Evolution with Estimation Distribution Algorithms as Search Engine". In: O. Castillo, P. Melin and J. Kacprzyk (eds.), Fuzzy Logic Augmentation of Neural and Optimization Algorithms: Theoretical Aspects and Real Applications, vol. 749, 2018, 169–177, DOI: 10.1007/978-3-319-71008-2_14.

[25] F. A. A. Motta, J. M. Freitas, F. R. Souza, H. S. Bernardino, I. L. Oliveira and H. J. C. Barbosa, "A Hybrid Approach of Grammar-based Genetic Programming and Differential Evolution for Symbolic Regression". In: Proceedings XIII Brazilian Congress on Computational Inteligence, 2018, 1–12, DOI: 10.21528/CBIC2017-110.

[26] F. A. A. Motta, J. M. de Freitas, F. R. de Souza, H. S. Bernardino, I. L. De Oliveira and H. J. C. Barbosa, "A Hybrid Grammar-Based Genetic Programming for Symbolic Regression Problems". In: 2018 IEEE Congress on Evolutionary Computation (CEC), 2018, 1–8, DOI: 10.1109/CEC.2018.8477826.

[27] D. A. Augusto and H. J. C. Barbosa, "Symbolic regression via genetic programming". In: Proceedings of Sixth Brazilian Symposium on Neural Networks, 2000, 173–178, DOI: 10.1109/SBRN.2000.889734.

[28] Q. Lu, J. Ren and Z. Wang, "Using Genetic Programming with Prior Formula Knowledge to Solve Symbolic Regression Problem", Computational Intelligence and Neuroscience, 2016, 1–17, DOI: 10.1155/2016/1021378.

[29] M. Nicolau and A. Agapitos, "Understanding Grammatical Evolution: Grammar Design". In: C. Ryan, M. O'Neill and J. Collins (eds.), Handbook of Grammatical Evolution, 2018, 23–53, DOI: 10.1007/978-3-319-78717-6_2.

[30] T. Nyathi and N. Pillay, "Comparison of a genetic algorithm to grammatical evolution for automated design of genetic programming classification algorithms", Expert Systems with Applications, vol. 104, 2018, 213–234, DOI: 10.1016/j.eswa.2018.03.030.

[31] C. R. Darwin, "On the origins of the species by means of natural selection, or the preservation of favoured races in the struggle for life", H. Milford, Oxford University Press, Cambridge, 1859.

[32] J. H. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence, The MIT Press, 1992, DOI: 10.7551/mitpress/1090.001.0001.

[33] D. Karaboga, C. Ozturk, N. Karaboga and B. Gorkemli, "Artificial bee colony programming for symbolic regression", Information Sciences, vol. 209, 2012, 1–15, DOI: 10.1016/j.ins.2012.05.002.