Studying the Properties of Structured Grammatical Evolution

ECOS-CISUC Technical Report TR2015/01

Nuno Lourenço Francisco B. Pereira Ernesto Costa

January 13, 2015

In this report we analyse the properties of Structured Grammatical Evolution (SGE), a new genotypic representation for Grammatical Evolution. In SGE, each gene is linked to a non-terminal of the grammar being used. By using this representation one ensures that the modification of a gene does not affect the derivation options of other non-terminals, thereby increasing locality. We use a set of static measures to characterise the interactions between the representation and variation operators and assess how they influence the interplay between the genotype-phenotype spaces. The performance of this new representation is compared with the standard GE on a set of benchmark problems. The results obtained, statistically validated, show that the adoption of SGE is beneficial.

1 Introduction

Evolutionary Algorithms (EA) are robust computational methods inspired by the principles of natural selection and genetics. They maintain a population of candidate solutions, which are evolved via variation operators. The individuals in the population continuously compete with each other to promote the appearance of better solutions. EAs have been successfully applied to a large domain of problems, from function optimization to the automatic evolution of computer programs. In particular, Genetic Programming (GP) is widely used by researchers and practitioners. Since its initial proposition, where syntax-trees were used to represent solutions, several representation variants of GP have been proposed. One of the first was based on grammars [31]. Grammars are extremely useful to represent construction rules and restrictions, and it did not came as a surprise when they were used to formalize constraints in GP.

At first, the grammar-based approaches were a small part of the GP world, but that changed with the appearance of Grammatical Evolution (GE) [18], which has become one of the most prominent GP methods. GE is different from other GP representations, since variation operators (i.e. recombination and mutation) are not performed on the actual

programs, but rather on a linear string with variable length. Thus a mapping process is needed to translate the string into programs, by relying on the rules that are specified in a grammar that restrict phenotypes to syntactically correct programs.GE is a state-of-art variant of GP, that obtained good results in difficult problems. However it has some known issues: low locality; and high levels of redundancy[12, 27, 30]. Concerning the former, it studies how variations performed in the genotype reflect on the phenotype. An algorithm has high locality if a small modification on the genotype results in a small modification on the phenotype. An EA that exhibits this kind of behavior is able to efficiently explore the neighborhood of the current solutions. If this condition is not satisfied, the exploration performed by the EA tends to resemble random search. Locality has been widely studied by researchers [9, 10, 23, 24]. A representation is said redundant when several different genotypes correspond to one phenotype. The usefulness of redundancy is an highly debated topic among researchers, but clearly excessive levels slows down evolution, thus decreasing the performance of the EAs [25].

The aim of this article is to propose an alternative genotypic representation, called Structured Grammatical Evolution (SGE). In SGE, the linear genotype is replaced by a structured one, where each gene is a list of integers that represent the possible derivation choices of non-terminals.

To study the properties (locality, redundancy) of this new representation, the framework of Raidl et al. [23] is adopted. They define a set of static measures to characterise the interactions between the representation and variation operators and assess how they influence the interplay between the genotype-phenotype spaces. The results allow us to gain insight on the redundancy and locality of GE and SGE. The analysis showed that SGE is less redundant and has better locality than GE.

Finally, we selected a set of benchmarks problems from [32], and perform a series of experiments to analyse the effectiveness of the SGE. We perform a comparison with a widely used GE engine, GEVA [16]. The results reveal that SGE obtained good results on the selected problems, as it was able to improve the results of GE. All the experiments were statistically validated.

We proceed by introducing GE and related research in Section 2. Section 3 details the SGE: the construction of the genotype, and how the performed the mapping between the genotype and the phenotype. Section 4 presents the redundancy and locality analysis of the new representation. It starts by giving the necessary definitions, and shows the results obtained. Section 5 details the experimental setup; the benchmarks where the approach was tested and the statistical tools used. Section 6 presents a discussion of the results obtained. Finally Section 7 gives some conclusions and points towards possible future work.

2 Background

2.1 Grammatical Evolution

GE is a form of grammar-based GP [15], proposed by O'Neill et al. [28, 18]. As in other GP variants, the goal of GE is to evolve executable programs/algorithmic strategies

ISSN: 0874-338X

that can exhibit good behavior when solving a given task. In early GP representations, programs were codified as syntax trees, and all the evolutionary operations were performed on the programs directly. On the contrary, in GE the variation operations are performed in linear strings. Initially the genotype was represented as a binary string, which was then translated to an integer string. However, practitioners usually skip the binary string and use the integer one directly, since it does not affect the performance of the algorithm [11].

A mapping process is required to map the string into an executable program, via productions rules of a grammar. A grammar is a tuple $G = (N, T, S, P)$, where N is a non-empty set of non-terminal symbols, T is a non-empty set of terminal symbols, S is an element of N called axiom, and P is a set of production rules of the form $A ::= \alpha$, with $A \in N$ and $\alpha \in (N \cup T)^*$. N and T are disjoint. Each grammar G defines a language $L(G)$, that is the set of all sequences of terminal symbols that can be derived from the axiom, also called words, that is $L(G) = \{w : S \overset{*}{\Rightarrow} w, w \in T^*\}$.

The key component in GE is the mapping from the genotype into the phenotype, i.e., the translation of the integer string to an executable program.
GE introduces a genotype to phenotype mapping in successive steps (see Fig. 1). Initially

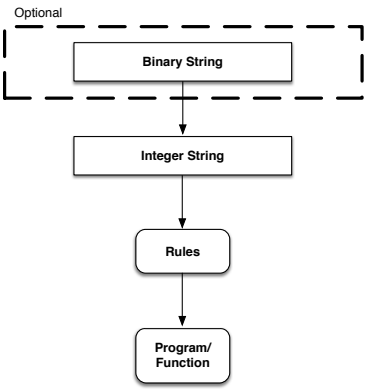


Figure 1: GE: from a binary string to an executable program (adapted from [15])

the genotype is a linear sequence of integers, each one called a codon. Then translation starts: a derivation tree (the phenotype) is obtained from the axiom of the grammar, and will be further decoded to an expression tree, the program. Each codon is used to determine the rule for a non-terminal symbol when it is expanded.
Suppose that we have the following production rule,

$$
\begin{aligned}
< expr > ::= & < expr >< op >< expr > && (0)\\
& |(< expr >) && (1)\\
& | < pre - op > (< expr >) && (2)\\
& | < var > && (3)
\end{aligned}
$$

where there are four options to rewrite the left-hand side symbol $< expr >$. In the beginning we have a syntactical form equal to the axiom $< expr >$. To rewrite the axiom one must choose which alternative will be used by taking the first integer and divide it by the number of options for $< expr >$. The remainder of that operation will indicate the option to be used. In the example above, assuming that the integer is 8, it follows that $8\%4 = 0$ and the axiom is rewritten in $< expr >< op >< expr >$. Then the the second integer is read, and the same method is used to the left most non-terminal of the derivation.

Sometimes the length of the string is not sufficient to complete the mapping into an executable program. In those cases the sequence can be reused repeatedly in a process known as wrapping. If after a pre-defined number of wrappings the mapping still cannot be performed, the process stops, and assigns the worst possible fitness value to the individual.

Since GE was proposed there has been a wide range of efforts to develop, improve, and extend it in various ways, including the genotype-phenotype mapping rule [12, 22], locality of the representation and search operators [27, 30], different search strategies [19, 21]. The most relevant contributions are briefly reviewed in the next sections.

2.2 Mapping

The standard mapping uses the modulo rule to map the integers to the correspondent symbol in a BNF grammar. However, over the years some alternatives have been proposed, in an attempt to improve the effectiveness of GE.

In [12], Keijzer et al. proposed an alternative called *bucket rule*. The goal is to remove the fact that each codon value always codes for the same production choice if the number of production rules are the same Instead, a mapping is done that allows the same codon value to code for different production choices, in order to remove the effect of rule definition ordering biasing the search. It is shown that by using the bucket rule the rule definition ordering bias is reduced.

In [22], O'Neill et al. introduced the Position Independent GE, or πGE, an alternative genotype-phenotype mapping. In the traditional GE mapping there is a positional dependency, as the derivation is always performed by expanding the leftmost terminal in the derivation tree. In πGE this dependency is removed by instead of representing each codon as a single value, as in standard GE, codons in πGE have two values: *nont* and *rule*. In this case *nont* indicates which should be the next non-terminal to be expanded. It does this using a similar approach to the original mapping rule:

$$NT = nont\%count \quad (1)$$

ISSN: 0874-338X

where NT is the next non-terminal to be consumed (counted from left to right of the remaining non-terminals), *nont* is the value present in the genotype, and *count* is the number of remaining non-terminals. The rule part of the codon pair, as in standard GE, selects which production rule should be applied to the selected non-terminal. This approach was validated in a set of benchmark problems and showed performance gains, when compared with standard GE.

Recently, Fagan and coworkers [5, 6] presented a study where they compared several mapping mechanisms. In addition to the standard GE mapping, which corresponds to a Depth-First expansion, three other were considered. The first was the Breadth-First expansion that maps all of the non-terminal symbols at each successive level of the derivation tree before moving on to the next deepest level. The second is the aforementioned πGE. Finally the third is a Random expansion mechanism, which randomly selects a non-terminal to expand among all of the non-terminals that have yet to be derived. The performance of the different mappings was measured across several benchmark problems, and the researchers concluded that πGE presents a viable alternative to the traditional GE mapping.

2.3 Locality

One of the first studies to address the locality on GE was the work by Rothlauf et al. [27]. The outcomes revealed that in 90% of the cases a mutation of a genotype (resulting in a genotypic neighbour) does not change the phenotype. This shows that GE suffers from high levels of redundancy, deriving from the many-to-one mapping, which allows multiple genotypic values to correspond to the same production rule. The second important outcome of this work is related with the remaining 10% of mutations. Specifically, when the genotype suffers a modification of one unit, changes of more than one unit occur at the phenotypic level. A unit of change in the phenotype corresponds to the tree edit distance [33], that includes deletion (delete a node from the derivation tree), insertion (insert a node in the derivation tree), and replacement (the label of a node is changed). Recently Thorhauer et al. [30] extended the previous work and compared the locality of the standard GE operators with standard GP. They analysed the locality on problems that used binary trees, by performing random walks through the search space and measuring the distance between parents and offspring. The outcome of the study confirms the results obtained in the previous study, and reinforces the idea that further investigation is needed to find alternative representations and associated operators which increase the locality in GE.

A study conducted by Byrne et al. [2, 3] proposes a distinction between two types of mutation in GE, based on their locality: a high-locality nodal mutation (which changes the value of node in the derivation tree) and a low-locality structural mutation (which changes the shape of the derivation tree). They perform an experimental study on the behaviour of these two operators and concluded that they have different goals. Whilst the nodal mutation increases the level of exploitation around promising solutions, the structural mutation promotes the exploration of the search space. The authors also acknowledge that a combination of the two types of mutation should be beneficial.

ISSN: 0874-338X

In Castle et al. [4], a study on the points of the recombination and mutation operator is presented. Their working hypothesis is that operations that occur in the beginning of the genotype are more destructive, as they may change completely the meaning of the following codons. The experimental analysis conducted confirms the working hypothesis, as they verified that the variation operators were more destructive when they occur at the first positions genotype.

Hugosson et al. [11] presents a study where they try to provide further understanding of GE, by investigating the impact of different genotypic representations, to determine whether certain representations, and associated operators, affect GE's effectiveness. Three genotypic representations are addressed: binary, gray coding, and integer. The experimental results revealed that none of the selected representations improved the GE's performance, when compared with each other. The authors also acknowledge that it does not imply that the representation is irrelevant. On the contrary it suggests that the utility of a specific representation cannot be isolated from the mapping process embedded in the grammar.

3 Structured Grammatical Evolution

In this section SGE, a new genotypic representation, is proposed. It aims to overcome two of the main issues associated with GE: low locality; and the high levels of redundancy introduced by module rule mapping.[4, 12, 27, 30].

3.1 New Representation

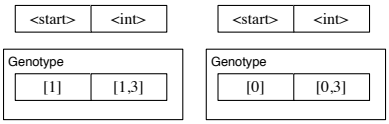
In SGE each gene comprises a list of integers and is linked to a specific non-terminal. The length of each list is determined by computing the maximum possible number of expansions of the connected non-terminal (see details below). This structure ensures that when a gene is modified, it does not affect the derivation options of other non-terminals, thus narrowing the number of changes that can occur at the phenotypic level.

The values that are inside the lists are directly associated with the number of possible expansion choices. Therefore, when performing the mapping it is possible to remove the modulo rule, thus reducing the redundancy associated with it. Consider the following set of production rules:

$$< start > ::= < int > \mid < int > * < int >$$
$$< int > ::= 1|2|3|4$$

There are two non-terminals $\{< start >, < int >\}$. The genotype is composed by two genes, where the first gene is linked to $< start >$, and the second to $< int >$. Then it is necessary to compute the length of the gene's lists by calculating the maximum number of expansions of a non-terminal. The $< start >$ symbol is expanded only once, as it is the grammar's axiom. The $< int >$ is expanded, at most, twice, because of the rule $< int > * < int >$. Thus the lists will have length 1 and 2. Finally, to fill them we count the number of possible derivation options, c_n, of each non-terminal and assign to

ISSN: 0874-338X



(a) Genotype 1 (b) Genotype 2

Figure 2: SGE: Example of two possible genotypes

each position of the list a random value from the interval $[0, c_N - 1]$. Considering the example above, the $< start >$ symbol has $c_N = 2$ and $< int >$ has $c_N = 4$. Two possible genotypes are depicted in Fig. 2.

The next step is to translate the genotype the phenotype. This process starts by expanding the axiom of the grammar, and then expanding the non-terminals in a left-first manner. Consider the example above, where the axiom is the non-terminal $< start >$. To expand it, we look at its gene within the genotype (Fig. 11a). The first unused integer of the list is 1, which selects the option $< int > * < int >$. The next symbol that needs to rewritten is $< int >$. Its first unused integer is 1, thus it is replaced by the option "2". Next the second $< int >$ is expanded. The first unused integer in the associated gene is 3, which dictates the option "4" should be selected. As there are no more symbols to expand, the process ends, and returns the phenotype: "2*4"

3.2 Pre-Processing

The first step to construct the genotype is to compute an upper bound for the number of times that a non-terminal can be expanded as it defines the size of the lists of each gene. Initially, we iterate through the productions belonging to the grammar, and record the maximum number of references to non-terminals that occur in each choice. At the same time we build a set that dictates a relation between non-terminals. Finally we determine recursively the number of times that, at most, a non-terminal will be expanded. The pseudocode to compute all these steps in detailed in Alg. 1. Consider the following set of production rules, with $< start >$ as the axiom:

$$\begin{aligned} &< start > ::= < line > \mid < line > / < line > \\ &< line > ::= < var > * < var > \\ &< var > ::= x1 \\ &\qquad\qquad \mid x2 \\ &\qquad\qquad \mid 1 \end{aligned}$$

Using the algorithm described above to compute the size of each gene, we would obtain: $< start >: 1, < line >: 2, < var >: 4$. Then we determine the values of c_n, i.e., the number of derivation choices, for each non-terminal: $< start >: 2, < line >: 1, < var >: 3$. With this it is now possible to build a genotype, as depicted in Fig. 3.

ISSN: 0874-338X

Algorithm 1 Pre-Processing: Compute an upper bond for the number of non-terminal expansions

```
function findReferences(nt, isReferencedBy, countReferencesByProd)
    r ← getTotalReferencesOfCurrentProduction(countReferencesByProd, nt)
    results ← []
    if nt = startSymbol then
        return 1
    end if
    for ref in isReferencedBy[nt] do
        result.append(findReferences(ref,isReferencedBy,countReferencesByProd))
    end for
    references ← references * max(result)
end function

function countReferences(isReferencedBy, countReferencesByProd)
    numberOfReferencesByNonTerminal ← {}
    for nt in nonTerminalsSet do
        numberOfReferencesByNonTerminal[nt] ←findReferences(nt, isReferencedBy,countReferencesByProd)
    end for
    return numberOfReferencesByNonTerminal
end function

function countReferencesToNonTerminals
    countReferencesInProduction ← {}
    isReferencedBy ← {}
    for nt in nonTerminalsSet do
        for production in grammar[nt] do
            count ← {}
            for option in production do
                if option ∈ nonTerminalsSet then
                    isReferencedBy[option] ← nt
                    count[option] ← count[option] + 1
                end if
            end for
        end for
        for key in count do
            countReferencesInProduction[key][nt] ← max(countReferencesInProduction[key][nt], count[key])
        end for
    end for
    return countReferences(isReferencedBy,countReferencesInProduction)
end function
```

ISSN: 0874-338X

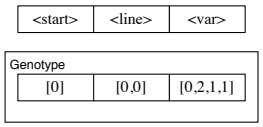


Figure 3: SGE genotype after the pre-processing

3.3 Recursive Grammars

To deal with recursion GE follows a "lazy" approach: it always tries to perform the translation into an executable program, until it runs out of integers. If that happens, GE assigns the worst fitness value possible to the individual.

SGE deals with recursion in a different way, as it follows a preemptive approach: a maximum level of recursion must be defined beforehand. Hence it is necessary to introduce a set of intermediate symbols that mimic the levels of the recursion tree. The following example is an excerpt of a grammar for symbolic regression problems:

$$
\begin{aligned}
&< start >::= < expr > \\
&< expr >::= < expr >< op >< expr > \mid < var > \\
&\quad < op >::= + \mid - \mid * \mid / \\
&\quad < var >::= x
\end{aligned}
$$

Looking at the grammar, we see that the $< expr >$ production is recursive. Therefore it needs to be rewritten. Assuming that 2 levels of recursion were defined it comes:

$$
\begin{aligned}
&< start >:= < expr > \\
&< expr >::= < expr\_lvl\_0 >< op >< expr\_lvl\_0 > \mid < var > \\
&< expr\_lvl\_0 >::= < expr\_lvl\_1 >< op >< expr\_lvl\_1 > \mid < var > \\
&< expr\_lvl\_1 >::= < var >< op >< var > \mid < var > \\
&\quad < op >::= + \mid - \mid * \mid / \\
&\quad < var >::= x
\end{aligned}
$$

While transforming the grammar we ensure two things: first, that all the symbols have the same probability of being selected after the transformation, because they are copied to each new added level; second, that there will be no invalid individuals, since the mapping process always finishes.

3.4 Genetic Operators

GE relies on standard operators to navigate the search space looking for promising solutions to the problem at hand. Two existing variation operators are adapted to work with the new representation.

ISSN: 0874-338X

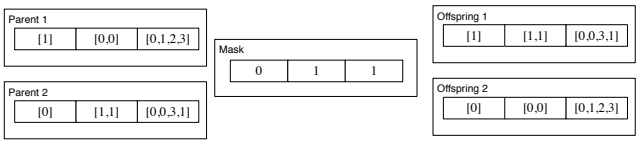


Figure 4: Example of the application of the recombination operator

Figure 5: Example of the application of the mutation operator

Recombination

The recombination operator proposed here is an adaptation of the Uniform crossover for binary representations [14]. It starts by creating a binary mask with the same length of the genotype. Then the offspring are created by swapping the parents genes based on the mask values. An example of how to use the recombination operator is showed in Fig. 4.

Mutation

The mutation proposed here is an adaptation of the integer flip operator. A gene is mutated by randomly selecting a position inside the list and replacing it by a random value from $[0, c_n - 1]$. If the generated value is equal to the current one, a new value is generated. An example of application of this operator is shown in Fig. 5.

4 Representation Analysis

This section presents an empirical study on the redundancy and locality of the proposed representation. The analysis adopts the framework proposed by Raidl et al. [23].

4.1 Basic Concepts

Spaces

When performing studies with EA usually two spaces are considered: the genotype space ϕ_g, and the phenotype space ϕ_p. The genetic operators are applied to solutions x, on ϕ_g. Each x is mapped to a solution X in the phenotype space via a mapping function f_g, such that $f_g(x) \rightarrow X$. Finally, the quality of each solution X is measured by a fitness function f, on ϕ_p: $f(X) : \phi_p \rightarrow \mathbb{R}$. Since SGE requires a mapping between genotype and phenotype distances for both spaces have to be defined.

ISSN: 0874-338X

Distances

The genotypic distance, d^g, is a variation of the Hamming for integer representations. Thus the distance between two arbitrary solutions $x, y \in \phi_g$ is defined as the total number of positions in which they differ.

The definition of a phenotypic distance d^p, between programs or expressions, is more challenging. However GE can describe its phenotypes as expressions trees making possible the usage of the edit distance to measure differences between two phenotypes. The edit distance between two trees calculate the number of minimal elemental operations that must be performed to transform one tree into the other. There are three elemental operations:

1. **Deletion**: A node is removed from the tree. The children of this node become children of its parent;
2. **Insertion**: A node is added;
3. **Replacement**: The label of a node is modified.

All operations have unitary cost. The tree edit measure has been used to measure similarities between trees in GP [1, 13, 20, 27].

Mutation Innovation

Let x be a solution in the genotype space, and x^m the solution resulting after applying mutation. The mutation innovation, MI, is the phenotypic distance between solution X and the mutated solution X^m,

$$MI = d^p(X, X^m) \tag{2}$$

MI describes how much the mutation operator modifies the phenotype of an individual. The distribution of the MI variable reflects several aspects concerning locality. The application of a locally strong operator implies that a small modification in the genotype results in a small modification of the phenotype, i.e., there will be a small phenotypic distance between the two involved solutions. On the contrary, operators with weak locality allow large jumps, which difficult the exploration of the search space.

When $MI = 0$ the mutation was no able to modify the phenotype of an individual. Hence, by counting the number of such events it is possible to analyse the redundancy of the representation.

Crossover Innovation

Crossover innovation measures the ability of the recombination operator to create offspring that are different from their parents. When using crossover, an offspring X^c is created from two parents X^{p1}, X^{p2}. The Crossover Innovation (CI) is the phenotypic distance between an offspring and is phenotypically closer parent,

$$CI = min(d^p(X^c, X^{p1}), d^p(X^c, X^{p2})) \tag{3}$$

ISSN: 0874-338X

$$N = \{expr, op, pre\_op, var\}$$
$$T = \{sin, cos, exp, log, +, -, *, /, x, 1.0, (,)\}$$
$$S = \{start\}$$

And the production set P is:

$$\begin{aligned}
< start > &::= < expr > \\
< expr > &::= < expr >< op >< expr > \\
&\quad |(< expr >) \\
&\quad | < pre_op > (< expr >) \\
&\quad | < var > \\
< op > &::= + | - | * |/ \\
< pre_op > &::= sin|cos|exp|log \\
< var > &::= x1|1.0
\end{aligned}$$

Figure 6: Symbolic regression grammar used to measure locality

Evidently, if $CI = 0$ implies that $X^c = X^{p1}$ or $X^c = X^{p2}$, thus crossover was not able to create a new solution. We expect that the value of CI is dependent on the values of the distances between the parents, i.e., $d^p(X^{p1}, X^{p2})$.

4.2 Results

The analysis conducted on the representation properties relies on symbolic regression grammars Fig. 6. Since the defined distances are based only on phenotypic distances, the definition of a target polynomial is not necessary. To compute the distances the dynamic programming approach proposed by Zhang et al. [33] is used.

Concerning numeric parameters, the values were selected to guarantee a fair comparison between the algorithms. Hence, the number of codons used in standard GE is set to 128, with 10 wraps, and the maximum level of recursion of SGE is set to 6.

Redundancy

To measure the redundancy of GE and SGE, we generated 10000 random individuals, X and applied a single mutation to each one, generating 10000 pairs (X, X^m). Then we measured the phenotypic distance between each pair and recorded the number of pairs where $d^p(X, X^m) = 0$. The experiments revealed that in GE about 90.3% of the mutations did not change the phenotype of an individual, while in SGE that percentage was only about 40%. This difference amid the two approaches is related to the way the mapping is performed: SGE uses a direct mapping between the values that are in

the lists (inside each gene), and the derivation option associated with the non-terminal. There is no need to use a complex mapping, using the modulo rule, as is the case of GE.

To complete the analysis we also measured the redundancy after a series of mutations. If during a run the levels of redundancy do not decrease it is an indication that the search space is not being properly explored, which may negatively impact the performance of the algorithm. Hence we gathered the previously generated 10000 individuals X, applied a series of $k = 20$ mutations to each one (generating 10000 pairs $(X, X^m)_j, j = 1...k$) and computed the resulting phenotypic distance d^p. For simplicity, the distances between the original solution and the successive mutated individuals are grouped in different sets. Given a certain d^p between two solutions, the set D_i to which d^p is assigned, is determined in the following way: $\{D0 : d^p = 0; D1 : d^p = 1; D2 : 1 < d^p \leq 5; D3 : 1 < d^p \leq 10; D4 : 10 < d^p \leq 20; D5 : 20 < d^p \leq 30; D6 : 30 < d^p \leq 40; D7 : 40 < d^p \leq 50; D8 : 50 < d^p \leq 60; D9 : 60 < d^p \leq 70; D10 : 70 < d^p \leq 80; D11 : 80 < d^p \leq 90; D12 : 90 < d^p \leq 100; D13 : d^p \geq 100; \}$. The values that were selected to construct the intervals seek to show the distribution of distances. In Fig. 7 we present the distribution of $d^p(X, X^m)$. For a given mutation step k, the gray-colored square represent the percentage of individuals, X^m, whose distance to the original solution, X, falls in the defined set D_i (darker squares indicate a higher percentage of individuals). For instance, for GE (Panel 7a) and the mutation step $k = 1$, about 90.3% of the individuals belong to the set $D0$, 4.5% belong to $D1$, 2.6% belong $D2$, and 1.1% belongs to $D3$, and so on. The plots depict that the redundancy decreases through the series of mutations. Nevertheless GE maintains a large number of individuals that do not suffer modifications. For example, when $k = 20$ about 50% of the mutations resulted in a $d^p(X, X^m) = 0$.

On the other hand looking at the redundancy displayed SGE (Panel 7b) we see that when $k = 1$ about 40% of the individuals belong to set $D0$, and as successive mutations are applied this value decreases, until when $k = 20$ the percentage is less then 10%. Moreover, the plot shows that the d^p distribution is more or less distributed across al the sets. This allows for a more adequate exploration of the search space.

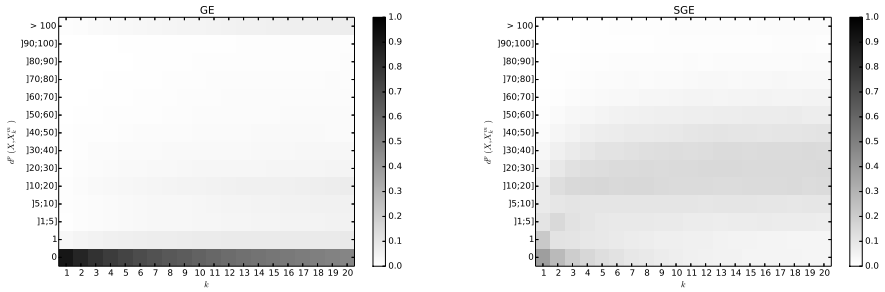
It is impossible to completely remove the SGE's redundancy, because of the way that the genotype is built. The length of the lists inside the genes is equal to maximum number of possible derivations, and when selecting a position to be mutated, it is not possible to know if it will be used in the translation process or not.

Locality

To calculate the MI, 10000 random individuals were generated, and then a sequence of $k = 20$ mutations was applied to each one of them. In each of 10000 mutation series, the distance is measured between the original individual and the solution generated after every mutation step.

The experiments conducted in these section focused on the individuals where $d^p > 0$. To ensure this, when a applying a mutation if it does not change the phenotype, we retract it, and apply another one, in a different gene. This process is repeated until $d^p > 0$.

ISSN: 0874-338X



(a) GE (b) SGE

Figure 7: Distribution of the percentage of individuals with $d^p = 0$ over a series $k = 20$ mutationsteps

In Fig.8 we present the distances between the original solutions and the individuals that were generated by iteratively applying mutations. The plot shows the results for each representation being analysed, and show the phenotypic distance between 2 randomly generated solutions (averaged over 10000 pairs of individuals). It is clear that experiments with SGE exhibit higher locality. Even after 20 steps, there is a relation between the mutated solution and the departure point. However it is import to note that the SGE's MI after $k = 15$ increases very slowly, which is an indication that the operator is not able to innovate. This may be an indication that mutation is not able to introduce new genetic material, which may cause the algorithm to become stuck.

By contrast GE depicts low locality. It is possible to see that after 13 mutation steps the distance between the original solution X, and the mutated one, X^m, is slightly great than the distance between two random solutions, which is an indication that at this point there is no relation between the original solution and the mutated one, and the algorithms starts to exhibit a random search behaviour.

The better locality of SGE is explained by the way the genotype is defined, where each gene is associated with a non-terminal. When a mutation occurs it changes the derivation option of that non-terminal without affecting others, i.e., they continue to represent the same derivation options. On the other hand, in GE, when a mutation occurs it may alter the derivation options of one or more non-terminals.

To compute the CI, a parent X^{p1} was randomly created and kept unchanged throughout all the experiments. The second parent, X^{p2}, was obtained from X^{p1} by the application of $k > 0$ successive mutations. In the experiments conducted, CI was measured after $k = 1, 2, 3, 4, ...20$ successive mutation steps.

The empirical results obtained revealed that in 77.1% and 48.7% the $CI = 0$ for GE and SGE, respectively, i.e., the recombination operator generated an individual that was equal to one of the parents. These results suggest that one must ensure that the population is diverse, else the algorithm may suffer a deterioration on the performance. When crossover is able to generate an offspring different from the parents, it has fairly degree of innovation, as Fig. 9 depicts.

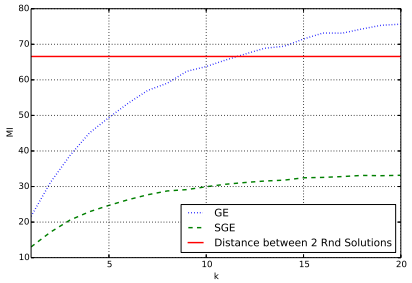


Figure 8: Distribution of the Mutation Innovation for a series of $k = 20$ mutation steps

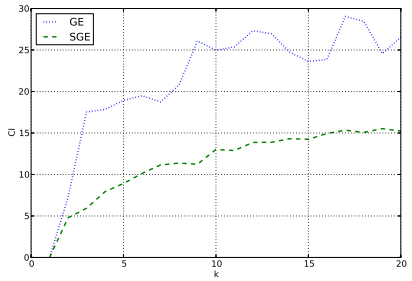
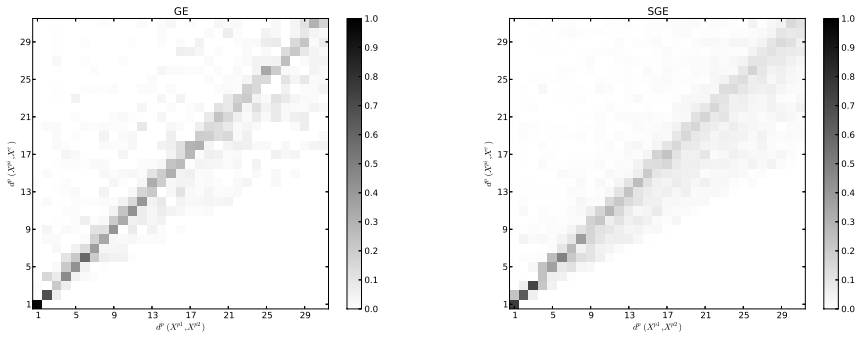


Figure 9: Distribution of the Crossover Innovation for a series of $k = 20$ mutation steps

CI presents a relation between the genotypic distance of the parents and the phenotypic distance of the generated offspring. Nonetheless it is also important to look at the relation between the phenotypic distance of the parents and offspring. A good recombination operator should be designed in a way that $max(d^p(X^{p1}, X^c), d^p(X^{p2}, X^c)) \leq d^p(X^{p1}, X^{p2})$, where X^{p1}, X^{p2} are the two parents, and X^c is the offspring. In other words, the offspring should be generated between the two parents, to guarantee that the search space is properly explored. This design principle is not new [26], and has been recently denominated as *geometry of search operators* [17].

In Fig. 10 we plot the distribution of distances between offspring and parents. For clarity reasons we only show results for phenotypic distances less than 30. The gray-colored squares indicate, for a given distance $d^p(X^{p1}, X^{p2})$, the percentage of offspring whose distance to one of their parents is $d^p(X^{pi}, X^c)$. Darker squares indicate an higher percentage. The values in the main diagonal mean that the individual is equal one of its parents. We are interested in the offspring that have distances in the upper left triangle, which are the individuals that do not verify the condition $d^p(X^{pi}, X^c) \leq d^p(X^{p1}, X^{p2})$. Considering a $d^p(X^{p1}, X^{p2} = 16$, about 40.1% of the GE (about 34% of SGE) offspring have a $d^p(X^{pi}, X^c) > 16$. These values indicate that a large of offspring generated by the crossover are not between its parents, which may negatively impact the performance of both GE and SGE.



(a) GE (b) SGE

Figure 10: Distribution of the offspring distances to its distance parent

5 Experimental Analysis

To validate the SGE a set of benchmark problems were selected following the guidelines proposed by White et al. [32]. Four problems were selected: quartic polynomial regression, harmonic curve regression, drug toxicity prediction, and the Santa Fe Ant Trail.

5.1 Problems Description

Symbolic Regression

Symbolic regression is one of the most commonly used benchmark. The goal is to find an expression that fits a certain polynomial. For our experiments we selected three symbolic regressions functions.

Quartic Polynomial The goal is to find a mathematical expression that can approximate the polynomial defined as

$$x^4 + x^3 + x^2 + x \tag{4}$$

with 20 data points in the interval $[-1, +1]$. The set of non-terminals is defined as $N = \{expr, op, pre\_op, var\}$, and the set of terminals is defined as $T = \{sin, cos, exp, log, +, -, *, /, x, 1.0, (,)\}$. The fitness for this problem is given by the sum of the absolute error between the evolved

and target functions. The production used to evolved polynomials is:

$$
\begin{aligned}
< start > ::= & < expr > \\
< expr > ::= & < expr >< op >< expr > \\
& |(< expr >) \\
& | < pre\_op > (< expr >) \\
& | < var > \\
< op > ::= & + | - | * | / \\
< pre\_op > ::= & sin|cos|exp|log \\
< var > ::= & x|1.0
\end{aligned}
$$

Harmonic Number The aim of this problem is to approximate the series defined by

$$\sum_{i}^{x} \frac{1}{i} \tag{5}$$

where $x \in [1, 50]$. This problem is interesting because it also requires an extrapolation, and not only and interpolation. After the evolutionary search the best solution is tested for generalisation over the interval $x \in [1, 120]$. The non-terminal set is composed by $N = \{expr, op, pre\_op, var\}$, and the terminal set $T = \{+, *, inverse, -i, sqrt, x\}$, where $inverse$ is $1/f(x)$. The production set for the harmonic number is defined as:

$$
\begin{aligned}
< start > ::= & < expr > \\
< expr > ::= & < expr >< op >< expr > \\
& |(< expr >) \\
& | < pre\_op > (< expr >) \\
& | < var > \\
< op > ::= & + |* \\
< pre\_op > ::= & + | - |inverse|sqrt \\
< var > ::= & x
\end{aligned}
$$

Drug Toxicity This is a real multidimensional symbolic regression problem from pharmacokinetics domain [29]. The aim of this application is to predict, the median lethal dose, known as LD50 of a set of candidate drug compounds. LD50 refers to the amount of compound required to kill 50% of the considered test organisms. The dataset consists of 234 instances, where each instance is a vector of 627 elements (626 molecular descriptor values identifying a drug, followed by the known LD50 for that drug). This data set

ISSN: 0874-338X

is known to be hard to model. The production set P for this problem is defined as:

$$\begin{aligned}
&< start >::= < expr > \\
&< expr >::= < expr >< op >< expr > \\
&\quad |(< expr >) \\
&\quad | < var > \\
&< op >::= + | - | * |/ \\
&< var >::= x| - 1.0| - 0.5|0.0|0.5|1.0
\end{aligned}$$

Artificial Ant

The goal in the artificial ant problem is to evolve a strategy that an agent will follow to collect food along a trail in a bi-dimensional toroidal grid. The ant will be placed at the upper left corner facing east, in a grid of 32x32 cells where there is a specific trail of food pellets with a few gaps in between, in this case the Santa Fe Ant Trail. The ant can turn left, right, move one square forward, might also look ahead one square in the direction it is facing to see if that square contains any piece of food. The set of non-terminals for this problem is $N = \{code, line, op\}$; the set of terminals is $T = \{if, ant.sense\_food(), else, ant.turn\_left(), ant.turn\_right(), ant.move\_forward(), :\}$ and the production P is:

$$\begin{aligned}
&< start >::= < code > \\
&< code >::= < line > \\
&\quad | < code > \\
&\quad < line > \\
&< line >::= if\ ant.sense_food() : \\
&\quad < line > \\
&\quad else : \\
&\quad < line > \\
&\quad | < op > \\
&< op >::= ant.turn_left() \\
&\quad | ant.turn_right() \\
&\quad | ant.move_forward()
\end{aligned}$$

5.2 Parameters

The GEVA implementation of GE was selected as baseline of comparison for our experiments. It is an open-source implementation of Grammatical Evolution, in JAVA, and is developed and maintained by O'Neill et al. [16].

ISSN: 0874-338X

Table 1: Parameter values for the Experimental Analysis

Parameter	GEVA	SGE-EA
Initial Population	500	
Recombination rate	0.9	
Mutation rate	0.02	
Replacement	Steady-State with a generation gap of 0.9	
Selection	Tournament with size 3	
Generations	50	
Recombination Operator	Single Point Crossover	SGE Uniform Crossover
Mutation Operator	Integer Flip Mutation	SGE Integer Flip Mutation
Genotype Size	128	-
Wraps	3	-
Maximum Level of Recursion	-	6

SGE uses a search engine based on GEVA. There are, however, some slightly changes: the set of variation operators used; and the definition of a maximum level of recursion. The parameters for both SGE and GEVA are defined in Table 1.

5.3 Statistical Validation

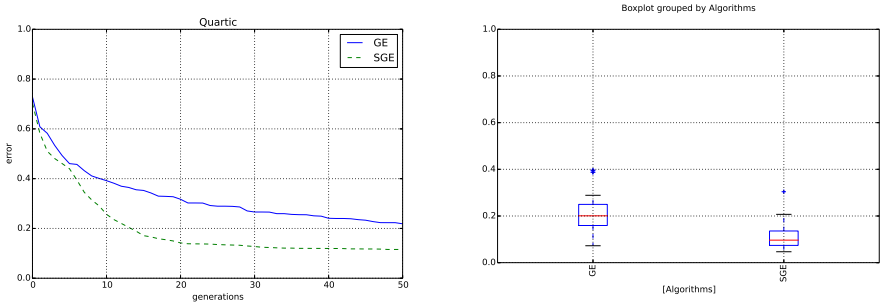
When comparing SGE with GE a statistical analysis was done to assess if there were differences in the means and, if that was the case, how relevant they were. Since the samples do not follow a normal distribution, the analysis was performed using non-parametric tests. Moreover, and as comparison the comparison is conducted between two unrelated groups the Mann-Whitney test, at a $\alpha = 0.05$ level of significance, was selected. The hypothesis defined for comparing the two approaches were:

- $H_0 : u_1 = u_2$ - means of the algorithms are equal, as the null hypothesis.
- $H_1 : u_1 \neq u_2$ - means of the algorithms are not equal, as the alternative hypothesis.

When a statistical test was applied we checked if the p-value $< \alpha$. If so, there was evidence to reject the null hypothesis H_0, and we could accept the alternative H_1. On the other hand, if the p-value $\geq \alpha$ there was not enough evidence to reject H_0, i.e., the means were not significantly different.

In the cases the null hypothesis was rejected we compute the effect size r [7], to determine how large the differences are. For clarity, we used the following notation: a +++ sign indicates that the effect size is large ($r > 0.5$). A ++ sign indicates that the effect size is medium ($0.3 < r < 0.5$), whereas a + identifies a small effect size ($0.1 < r < 0.3$). Finally, a $\sim$ indicates that no statistical differences between the algorithms were found.

ISSN: 0874-338X



(a) Quartic Polynomial Mean Best Fitness (b) Quartic Polynomial Box Plot

Figure 11: Mean Best Fitness plots for the Quartic polynomial (lower is better)

5.4 Results

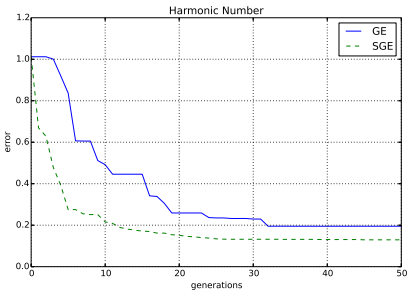
Fig. 11 shows the evolution of the Mean Best Fitness(MBF) during the 50 generations for the quartic (Fig. 11a) polynomial. Looking at the results, it is possible to see that for the quartic problem (Fig. 11a) SGE is successful. It is able to achieve better fitness values faster than standard GE and it can also reach significantly better values. A closer inspection of the plot shows that the fitness of the initial population of both approaches is slightly different. This can be explained by the fact that SGE does not create invalid individuals, while GE does. When this happens, it assigns the worst possible fitness (usually a large value for minimisation problems). In the case of the harmonic problem, about 35% of the individuals are invalids. The boxplots of Fig. 11b help to clarify the results attained.

In the harmonic number problem we have to analyse the results obtained in a training set (i.e., interpolation) and in a testing set (i.e. extrapolation). Fig. 12 shows the optimisation results for each of the steps. Looking at panel 12a we can see that the behaviour is very similar to the one observe in the quartic polynomial. SGE can rapidly find good solutions, while GE requires more time. In what concern quality of the best solutions, it is possible to see by the boxplot that the new representation is able to find solutions with smaller errors than GE.

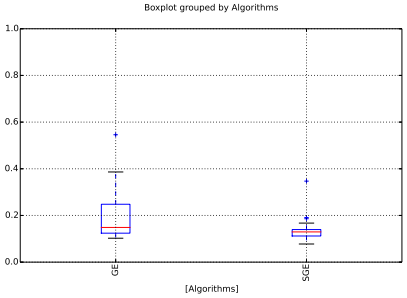
Regarding the test phase (Fig. 12c), it is possible to see the same trend as in training: SGE rapidly decreases the testing error, but after 11 generations it is not able to find a better solution.

The last regression problem is the drug toxicity (LD50). In this problem we follow a method similar to the harmonic number problem: training and testing. The dataset is divided in two equal partitions. The first partition is used in the evolutionary search to find a regression model. The second part is used to validate the best-evolved model. Since the problem is more difficult than the previous function, the number of generations is increased to 200. The results are quite similar to the harmonic problem. Fig. 13a shows that for the training phase, SGE obtains good quality faster than GE.

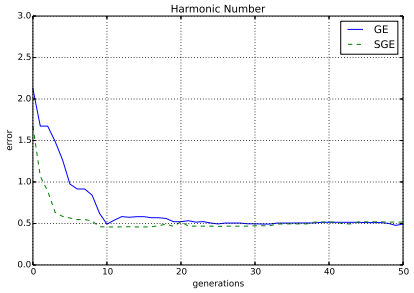
Concerning testing, SGE and GE have both the same error during most of the evolu-



(a) Training Mean Best Fitness



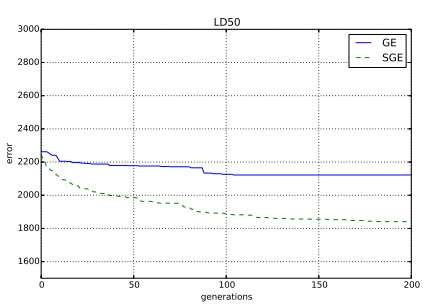
(b) Training BoxPlot



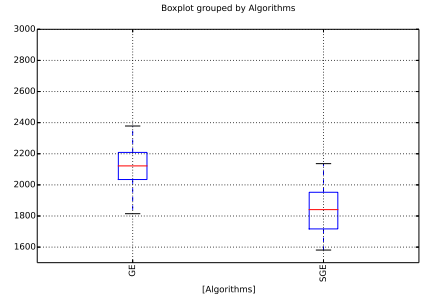
(c) Testing Mean Best Fitness

Figure 12: Mean Best Fitness plots of training and testing for the Harmonic Number

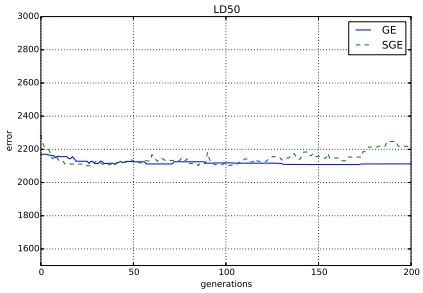
ISSN: 0874-338X



(a) Training Mean Best Fitness



(b) Training BoxPlot



(c) Testing Mean Best Fitness

Figure 13: Mean Best Fitness plots of interpolation and extrapolation for the LD50 problem

tionary run. Although at the end of the run SGE's testing error suffers a deterioration (Fig. 13c). This means that the patterns that SGE is learning do not apply to unseen cases, which leads to overffiting of the training data. Moreover, the methods present in the literature to control overffiting are not able to obtain a testing error less than 2000. Thus it is expected that, as the training error falls under 2000, the testing error increases [8].

The final benchmark used is the Santa Fe Ant Trail. In Fig. 14 it is possible to see that SGE clearly obtain better results than GE. It is also possible to see that after 25 generations (half of the total number of generations), the new representation already found a solution that can effectively solve the problem. In terms of success rate, in 30 out of the 30 runs, SGE was able to find solutions that allowed the ant to eat all the food pieces in the board.

Finally, SGE and GE were compared using the statistical tools previously described. The results are presented in the column Statistical Validation of Table 2. They reveal that SGE provides statistical significant improvements on the traditional GE. We present the p-values obtained, to clarify the magnitude of the differences. The highest p-value is the harmonic value, and it still is far from the $\alpha = 0.05$ that we selected as level of significance. We also computed the effect sizes, to assess how large the differences were.

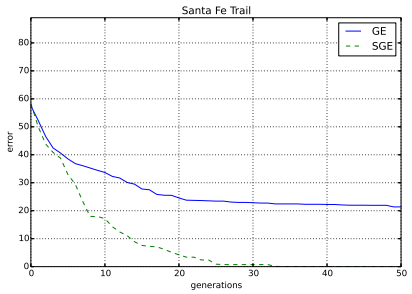


Figure 14: Mean Best Fitness plots for the Santa Fe Ant Trail

Table 2: Training Results: Mean Best Fitness and Standard Deviation over 30 runs

	Training			
			Statistical Validation	
Problem	GE	SGE	p-value	Effect Size
Harmonic	0.20 ($\pm$ 0.11)	0.13($\pm$ 0.05)	$7.21 * 10^{-3}$	++
Drug Toxicity	2115.7 ($\pm$ 143.97)	1841.69 ($\pm$ 160.33)	$1.37 * 10^{-8}$	+++
Santa Fe Ant Trail	21.40 ($\pm$ 12.40)	0.00 ($\pm$ 0.00)	$9.21 * 10^{-14}$	+++

The only problem were the effect size was medium ($0.3 < r < 0.5$) was the harmonic number. In all other problems the effect size was large.

Regarding generalization, the same statistical tools were applied, but no meaningfull differences were found.

6 Discussion

The main contribution of this article is the proposal of a structured representation for GE.

The first difference between the SGE and GE is the absence of invalid individuals. In GE, when it is impossible to complete a translation of a program, we define the individual as invalid, and give it the worst possible fitness. These appear mostly when we use a recursive grammar. The new representation avoids invalids by defining, before hand, a maximum number of recursion levels.

SGE also reduces the number of numeric parameters that one must specify before an evolutionary run. While in GE one has to define the codon size, the number of wrappings, and the genome size, in the new representation we only have to decide on the maximum level of recursion. The other parameters are obtained in the pre-processing step.

Another aspect that has been discussed by several researches is the redundancy of GE, and whether it is useful or not. We performed an empirical analysis on the redundancy of both representations, and saw the new representation had $\sim$ 50% less redundancy than traditional GE. The fact that the new representation has less redundancy does not

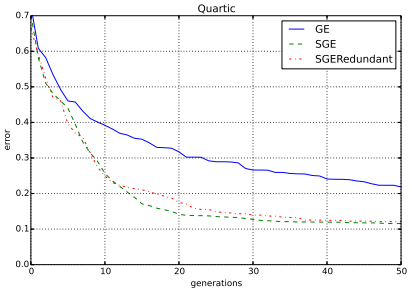


Figure 15: SGE compared with an alternative version with increase redundancy

affect the effectiveness of the algorithm, as we can see by the results on the benchmark problems.

An additional test was conducted with a alternative version of the SGE, where the genotype-phenotype mapping used the modulo rule, thus increasing the levels of redundancy. The experiments revealed that this modification increase the levels of redundancy of SGE to 60%. Fig. 15 depicts the results of this modification in a regression problem (results are similar for the other problems). The results reveal that there are no advantages of removing the redundancy introduced by the modulo rule.

Some studies revealed that GE had problems with locality, since that a small change in the genotype resulted in a large change of the phenotype. In this work we analysed the locality of both GE and SGE. We relied on innovation measures (number of changes) of the variation operators. The results obtained confirmed the GE locality problems, and showed that SGE has a better locality.

The main strength of SGE is the higher locality that it exhibits. The additional test conducted in Fig. 15 shows that redundancy does not severely affect de performance of the algorithm.

7 Conclusions

Grammatical Evolution (GE) is a type of grammar-based GP. Contrary to other GP approaches that modify the programs directly, GE applies the evolutionary operations on a binary/integer integer string. A mapping process is required to map the string to an executable program, via productions rules of a grammar. Despite being widely used, it has a set of drawbacks, mainly the low locality, and the high levels of redundancy.

To overcome these issues we propose a new genotypic representation, called SGE. SGE builds a structured genotype based on the grammar being used. The construction of the genotype is performed in two steps: First we check for recursive productions, and rewrite these productions, by adding additional non-terminal symbols to simulate the recursion levels. The total number of recursion levels is an external parameter defined by the user. Secondly the maximum number of non-terminals expansions is computed. The genotype is constructed based on these computations: each gene is associated with a non-terminal,

ISSN: 0874-338X

and has a list of integers that corresponds to the option that should be selected at a certain time. The mapping process is performed by reading each non-terminal, looking at its gene and selecting the first non-used option. Along with the proposal of the genotypic representation two variation operators were adapted: recombination and mutation. SGE can be situated as an hybrid between GE and the traditional grammar-based GP.

An analysis on the redundancy, i.e., on the number of mutations that did not change the phenotype was performed. We confirmed the results already measure for GE, where 90% of the mutations did not cause phenotypic modifications. On the contrary SGE evinces much less redundancy since in only 40% the cases, mutations did not alter the phenotype.

The locality of SGE was studied, using the framework prosed by Raidl et al. [23], which is based on the innovation of the operators. These measures showed that after an certain number of accumulated mutations, the difference between the initial solution and the final one are smaller in SGE than in GE, which is an indication of better locality.

Finally we selected a set of benchmarks to test our approach. We selected some difficult problem scenarios and the results were encouraging, as the new approach obtained good results, specially when compared with GE. The new representation can also delivered good quality solutions in a very low number of generations.

Concerning future we want to analyse the behaviour of the new representation in the domain of the Hyper-Heuristics, as it can deliver good results faster. We also intend to study and propose new operators that may increase the effectiveness of the approach.

References

[1] Markus Brameier and Wolfgang Banzhaf. Explicit control of diversity and effective variation distance in linear genetic programming. In JamesA. Foster, Evelyne Lutton, Julian Miller, Conor Ryan, and Andrea Tettamanzi, editors, *Genetic Programming*, volume 2278 of *Lecture Notes in Computer Science*, pages 37–49. Springer Berlin Heidelberg, 2002.

[2] Jonathan Byrne, Michael O'Neill, and Anthony Brabazon. Structural and nodal mutation in grammatical evolution. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, pages 1881–1882, New York, NY, USA, 2009. ACM.

[3] Jonathan Byrne, Michael ONeill, James McDermott, and Anthony Brabazon. An analysis of the behaviour of mutation in grammatical evolution. In AnnaIsabel Esparcia-Alczar, Anik Ekrt, Sara Silva, Stephen Dignum, and A.ima Uyar, editors, *Genetic Programming*, volume 6021 of *Lecture Notes in Computer Science*, pages 14–25. Springer Berlin Heidelberg, 2010.

[4] Tom Castle and ColinG. Johnson. Positional effect of crossover and mutation in grammatical evolution. In AnnaIsabel Esparcia-Alczar, Anik Ekrt, Sara Silva, Stephen Dignum, and A.ima Uyar, editors, *Genetic Programming*, volume 6021

of *Lecture Notes in Computer Science*, pages 26–37. Springer Berlin Heidelberg, 2010.

[5] David Fagan. An analysis of genotype-phenotype mapping in grammatical evolution, 2014.

[6] David Fagan, Michael ONeill, Edgar Galvn-Lpez, Anthony Brabazon, and Sean McGarraghy. An analysis of genotype-phenotype maps in grammatical evolution. volume 6021 of *Lecture Notes in Computer Science*, pages 62–73, 2010.

[7] Andy P Field. *How to Design and Report Experiments*. SAGE, 2003.

[8] Ivo Gonçalves and Sara Silva. Balancing learning and overfitting in genetic programming with interleaved sampling of training data. In *Proceedings of the 16th European Conference on Genetic Programming, EuroGP 2013*, volume 7831 of *LNCS*, pages 73–84, Vienna, Austria, 2013. Springer Verlag.

[9] Jens Gottlieb and Christoph Eckert. A comparison of two representations for the fixed charge transportation problem. In Marc Schoenauer, Kalyanmoy Deb, Gnther Rudolph, Xin Yao, Evelyne Lutton, JuanJulian Merelo, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature PPSN VI*, volume 1917 of *Lecture Notes in Computer Science*, pages 345–354. Springer Berlin Heidelberg, 2000.

[10] Jens Gottlieb and GntherR. Raidl. Characterizing locality in decoder-based eas for the multidimensional knapsack problem. In Cyril Fonlupt, Jin-Kao Hao, Evelyne Lutton, Marc Schoenauer, and Edmund Ronald, editors, *Artificial Evolution*, volume 1829 of *Lecture Notes in Computer Science*, pages 38–52. Springer Berlin Heidelberg, 2000.

[11] Jonatan Hugosson, Erik Hemberg, Anthony Brabazon, and Michael O'Neill. Genotype representations in grammatical evolution. *Applied Soft Computing*, 10(1):36 – 43, 2010.

[12] Maarten Keijzer, Michael ONeill, Conor Ryan, and Mike Cattolico. Grammatical evolution rules: The mod and the bucket rule. In JamesA. Foster, Evelyne Lutton, Julian Miller, Conor Ryan, and Andrea Tettamanzi, editors, *Genetic Programming*, volume 2278 of *Lecture Notes in Computer Science*, pages 123–130. Springer Berlin Heidelberg, 2002.

[13] Robert E. Keller and Wolfgang Banzhaf. Genetic programming using genotype-phenotype mapping from linear genomes into linear phenotypes. In *Proceedings of the 1st Annual Conference on Genetic Programming*, pages 116–122, Cambridge, MA, USA, 1996. MIT Press.

[14] Sean Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

ISSN: 0874-338X

[15] Robert I McKay, Nguyen Xuan Hoai, Peter Alexander Whigham, Yin Shan, and Michael O'Neill. Grammar-based Genetic Programming: a survey. *Genetic Programming and Evolvable Machines*, 11(3-4), September 2010.

[16] Conor Gilligan Eliott Bartley James McDermott Anthony Brabazon Michael ONeill, Erik Hemberg. Geva - grammatical evolution in java (v 2.0). Technical report, 2008.

[17] Alberto Moraglio and Riccardo Poli. Topological interpretation of crossover. In Kalyanmoy Deb, editor, *Genetic and Evolutionary Computation* , volume 3102 of *Lecture Notes in Computer Science*, pages 1377–1388. Springer Berlin Heidelberg, 2004.

[18] M. O'Neill and C. Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers, Norwell, MA, USA, 2003.

[19] Michael O'Neill and Anthony Brabazon. Grammatical differential evolution. In Hamid R. Arabnia, editor, *Proceedings of the 2006 International Conference on Artificial Intelligence, ICAI 2006*, volume 1, pages 231–236, Las Vegas, Nevada, USA, June 26-29 2006. CSREA Press.

[20] U.-M. O'Reilly. Using a distance metric on genetic programs to understand genetic operators. In *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on*, volume 5, pages 4092–4097 vol.5, Oct 1997.

[21] Michael ONeill and Anthony Brabazon. Grammatical swarm: The generation of programs by social programming. *Natural Computing*, 5(4):443–462, 2006.

[22] Michael ONeill, Anthony Brabazon, Miguel Nicolau, SeanMc Garraghy, and Peter Keenan. pigrammatical evolution. In *Genetic and Evolutionary Computation GECCO 2004*, volume 3103, pages 617–629. Springer Berlin Heidelberg, 2004.

[23] Günther R. Raidl and Jens R. Gottlieb. Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem. *Evol. Comput.*, 13(4):441–475, December 2005.

[24] Franz Rothlauf. On the locality of representations. In Erick Cant-Paz, JamesA. Foster, Kalyanmoy Deb, LawrenceDavid Davis, Rajkumar Roy, Una-May OReilly, Hans-Georg Beyer, Russell Standish, Graham Kendall, Stewart Wilson, Mark Harman, Joachim Wegener, Dipankar Dasgupta, MitchA. Potter, AlanC. Schultz, KathrynA. Dowsland, Natasha Jonoska, and Julian Miller, editors, *Genetic and Evolutionary Computation GECCO 2003*, volume 2724 of *Lecture Notes in Computer Science*, pages 1608–1609. Springer Berlin Heidelberg, 2003.

[25] Franz Rothlauf. *Representations for genetic and evolutionary algorithms*. Springer, 2006.

ISSN: 0874-338X

[26] Franz Rothlauf. *Design of modern heuristics: principles and application*. Springer, 2011.

[27] Franz Rothlauf and Marie Oetzel. On the locality of grammatical evolution. In *Proceedings of the 9th European Conference on Genetic Programming*, EuroGP'06, pages 320–330, Berlin, Heidelberg, 2006. Springer-Verlag.

[28] Conor Ryan, JJ Collins, and MichaelO Neill. Grammatical evolution: Evolving programs for an arbitrary language. In Wolfgang Banzhaf, Riccardo Poli, Marc Schoenauer, and TerenceC. Fogarty, editors, *Genetic Programming*, volume 1391 of *Lecture Notes in Computer Science*, pages 83–96. Springer Berlin Heidelberg, 1998.

[29] Sara Silva and Leonardo Vanneschi. State-of-the-art genetic programming for predicting human oral bioavailability of drugs. 74:165–173, 2010.

[30] Ann Thorhauer and Franz Rothlauf. On the locality of standard search operators in grammatical evolution. In Thomas Bartz-Beielstein, Juergen Brank, and Jim Smith, editors, *13th International Conference on Parallel Problem Solving from Nature*, Ljubljana, Slovenia, 13-17 September 2014.

[31] P.A. Whigham and Department Of Computer Science. Grammatically-based genetic programming, 1995.

[32] DavidR. White, James McDermott, Mauro Castelli, Luca Manzoni, BrianW. Goldman, Gabriel Kronberger, Wojciech Jakowski, Una-May OReilly, and Sean Luke. Better gp benchmarks: community survey results and proposals. *Genetic Programming and Evolvable Machines*, 14(1):3–29, 2013.

[33] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, December 1989.

ISSN: 0874-338X