Weighted Hierarchical Grammatical Evolution

Alberto Bartoli, Mauro Castelli, and Eric Medvet

Abstract—**Grammatical Evolution (GE) is one of the most widespread techniques in evolutionary computation. Genotypes in GE are bit strings while phenotypes are strings of a language defined by a user-provided context-free grammar (CFG). In this work, we propose a novel procedure for mapping genotypes to phenotypes that we call Weighted Hierarchical GE (WHGE). WHGE imposes a form of hierarchy on the genotype and encodes grammar symbols with a varying number of bits based on the relative expressive power of those symbols. WHGE does not impose any constraint on the overall GE framework, in particular, WHGE may handle recursive grammars, uses the classical genetic operators, and does not need to define any bound in advance on the size of phenotypes.**

We assessed experimentally our proposal in depth on a set of challenging and carefully selected benchmarks, comparing the results to the standard GE framework as well as to two of the most significant enhancements proposed in the literature: Position-independent GE and Structured GE. Our results show that WHGE delivers very good results in terms of fitness as well as in terms of the properties of the genotype-phenotype mapping procedure.

I. Introduction

Grammatical Evolution (GE) [1], [2] is a variant of Genetic Programming (GP) [3] that can evolve complete programs in any language. This capability directly derives from the genotype-phenotype *mapping* of GE: genotypes in GE are either bit or integer strings mapped to strings of a language defined by a user-provided context-free grammar (CFG) [4], [5], [6], [7]. Internally, the functionality of GE follows standard Evolutionary Algorithm (EA) approaches. This mechanism relieves the user from the burden of adapting the internals of the EA to his specific problem, hence favoring GE usage in a wide range of applications: e.g., automatic composition of music [8], road traffic rules synthesis [9], generation of string similarity indexes suitable for text extraction [10], optimization of discrete planar truss [11], and even the design of other optimization algorithms [12].

The success and conceptual elegance of GE have stimulated a wealth of research in this area, including several proposals aimed at improving the framework effectiveness. The proposal in [13] could not demonstrate the ability to generate good solutions, whereas others were designed specifically for particular cases [14], [15]. Some proposals, however, have been significantly successful as their ability to deliver better solutions than the standard GE framework was demonstrated in a broad variety of benchmarks [16], [17]. *Position-independent GE (πGE)* modified the standard GE framework only in terms of a different genotype-phenotype mapping procedure [16], while the recent *Structure-independent GE (SGE)* advocated a more radical departure from the original framework, based on a different genotypic representation and novel genetic operators tailored to that representation [17]. In the event the user-provided grammar is recursive, SGE requires that the grammar be modified preliminarily and expressed in a non-recursive form, by means of a procedure also described in [17].

In this work we propose a novel variant of GE that we call *Weighted Hierarchical GE (WHGE)*. The only change with respect to the standard GE framework consists of a novel genotype-phenotype mapping procedure, in particular, WHGE may operate with standard genetic operators on user-provided grammars that may possibly be recursive. The derivation tree of the phenotype is constructed by imposing a form of *hierarchy* on the genotype: the genotype is (recursively) partitioned in several substrings, each that maps to a subtree of the derivation tree. Furthermore, genotype partitions are not of the same size: symbols are *weighted* based on their *expressive power*, that is, a symbol with many derivation options in the grammar will be given more genotype bits than a symbol with few derivation options.

We assessed our proposal experimentally in depth, on a number of challenging benchmark problems that we selected carefully based on the guidelines for the evaluation of Genetic Programming approaches proposed in [18], [19]. WHGE compares very favourably to GE, πGE, and SGE in terms of the fitness of the generated solutions.

We extended our assessment to the *evolvability* of each GE variant, i.e, the tendency of generating fitter individuals during the evolution, as well as to specific *properties* of the genotype-phenotype mapping procedures [20], [21], in particular, the tendency of generating individuals that cannot be mapped into a phenotype (*invalidity*), the tendency of mapping multiple different genotypes on the same phenotype (*degeneracy*), the tendency of mapping genotypic neighbors to phenotypic neighbors (*locality*) and the combined tendency of a genetic operator and a mapping procedure to lead to the same phenotype (*neutrality*) [22], [23], [24], [25], [26], [27], [28]. In this respect, we observed that WHGE tends to exhibit much better evolvability and degeneracy than the other variants, while SGE tends to have the best locality.

An earlier and very preliminary version of this work appeared in [29]. The present work extends the cited paper in several directions: a more detailed description of the proposed mapping and of its motivations, a much broader and deeper set of experiments including more benchmark problems and more competitors, as well as an analysis of the mapping properties.

The paper is organized as follows: Section II presents related work and outlines the working principles of the most

Alberto Bartoli and Eric Medvet are with the Department of Engineering and Architecture (DIA), University of Trieste, 34127 Trieste, Italy. (email: emedvet@units.it, bartoli.alberto@units.it)

Mauro Castelli is with the NOVA Information Management School (NOVA IMS), Universidade Nova de Lisboa, Campus de Campolide, 1070-312 Lisboa, Portugal. (email: mcastelli@novaims.unl.pt)

Manuscript received ...

⟨expr⟩ ::= (⟨expr⟩ ⟨op⟩ ⟨expr⟩) | ⟨num⟩ | ⟨var⟩
⟨op⟩ ::= + | - | * | /
⟨var⟩ ::= x | y
⟨num⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Fig. 1: A CFG in the Backus-Naur Form (BNF) for mathematical expressions. Following the usual convention, we specify the starting symbol implicitly as the non-terminal on the left side of the first rule.

significant existing mappings; Section III describes our WHGE proposal and explains the design motivations; Section IV describes the experimental assessment and discusses the corresponding results; Section V concludes the paper summarizing the main findings and suggests avenues for future work.

II. Related work: GE variants

The salient aspect of GE is its *genotype-phenotype mapping* procedure, which allows transforming a bit string (the genotype) in a program (the phenotype), i.e., a string of the language $\mathcal{L}(\mathcal{G})$ described by the context-free grammar (CFG) $\mathcal{G}$. The CFG is defined by the tuple (N, T, s_0, R), where N is the set of *non-terminal* symbols, T is the set of *terminal* symbols (with $T \cap N = \emptyset$), $s_0 \in N$ is the starting symbol, and R is the set of *production rules*. Figure 1 shows the production rules of an example CFG using the Backus-Naur Form (BNF): the starting symbol is $s_0 = \langle\text{expr}\rangle$ and the corresponding subset $R_{\langle\text{expr}\rangle}$ of production rules consists of three rules: ⟨expr⟩ → (⟨expr⟩ ⟨op⟩ ⟨expr⟩), ⟨expr⟩ → ⟨num⟩, and ⟨expr⟩ → ⟨var⟩. We call *derivation* the application of a production rule consisting in the replacement of the non-terminal symbol on the left-hand side of the production rule with the symbols on the right-hand side.

Before describing our WHGE proposal, we describe the standard GE procedure [1] and its most significant variants, *Position-independent GE (πGE)* [16], and the more recent proposal *Structured Grammatical Evolution (SGE)* [17]. These three frameworks are all used as baselines in our experimental evaluation of WHGE.

A. Standard GE mapping

In standard GE [1], the genotype is split into substrings of n consecutive bits which are then translated into integers using the natural binary encoding—each integer being called *codon*. The value of the parameter n is conventionally set to 8, but in some applications it has been set to the lowest value which is greater than or equal to the maximum number of production rules for a non-terminal of the grammar (e.g., [10]), with the aim of reducing degeneracy.

The procedure for mapping the input genotype g into a phenotype p is iterative and starts with $p = s_0$, a counter $i = 0$, and a counter $w = 0$. Then, the following steps are iterated—Figure 2 shows an example of execution.

1) The leftmost non-terminal s in p is derived using the j-th production rule in R_s (zero-based indexing). The value of j is set to $g_i \mod |R_s|$, i.e., the remainder of the division between the value g_i of the i-th codon (zero-based indexing) and the number $|R_s|$ of production rules for s.
2) The counter i is incremented; if it exceeds the number of codons $\frac{|g|}{n}$, then i is set to 0 and w is incremented.
3) If p contains at least one non-terminal, return to step 1, otherwise end.

$$\begin{aligned} g = {} & \text{11100111 11110000 10100001} & \\ & \text{01110001 01001101 00000111} & \text{(bits)} \\ = {} & \text{231 15 133 142 178 224} & \text{(integers)} \end{aligned}$$

i	g_i	$\lvert R_s \rvert$	j	w	Phenotype p
0	231	3	0	0	⟨**expr**⟩
1	15	3	0	0	(⟨**expr**⟩ ⟨op⟩ ⟨expr⟩)
2	133	3	1	0	((⟨**expr**⟩ ⟨op⟩ ⟨expr⟩) ⟨op⟩ ⟨expr⟩)
3	142	10	2	0	((⟨**num**⟩ ⟨op⟩ ⟨expr⟩) ⟨op⟩ ⟨expr⟩)
4	178	4	2	0	((2 ⟨**op**⟩ ⟨expr⟩) ⟨op⟩ ⟨expr⟩)
5	224	3	2	0	((2 * ⟨**expr**⟩) ⟨op⟩ ⟨expr⟩)
0	231	2	1	1	((2 * ⟨**var**⟩) ⟨op⟩ ⟨expr⟩)
1	15	4	3	1	((2 * y) ⟨**op**⟩ ⟨expr⟩)
2	133	3	1	1	((2 * y) / ⟨**expr**⟩)
3	142	10	2	1	((2 * y) / ⟨**num**⟩)
					((2 * y) / 2)

Fig. 2: Steps of the standard GE mapping procedure with a genotype g of 48 bits and the grammar of Figure 1. The rightmost column shows the phenotype p before the derivation of the highlighted non-terminal.

The reuse of the genotype which is triggered by the first condition at step 2 is called *wrapping*. A maximum of n_w wrappings are allowed; whenever all of them are executed, the mapping is aborted: the individual is then referred to as *invalid* or non-valid and conventionally associated with the worst possible fitness value [28]. Wrapping allows GE mapping to handle the case in which the genotype is consumed before the mapping is ended, i.e., when one or more non-terminals are still present in the phenotype. This case may occur in particular with complex or recursive grammars, the latter corresponding to languages containing non-finite strings which are, in facts, of great practical relevance.

B. πGE mapping

The mapping procedure of Position-independent GE (πGE) [16] is based on the standard GE procedure: however, instead of deriving the leftmost non-terminal, the procedure derives a non-terminal which is chosen using the genotype itself. According to the authors, this modification should decouple the position at which a production rule is applied from the choice of the production rule to apply, the aim of the decoupling being to favor the arising of useful building blocks (i.e., short subsequences) in the genotype. Nevertheless, in their experiments, the authors did not find any significant evidence of the desired effect. On the other hand, it has been

$$\begin{aligned} g &= \texttt{11100111 11110000 10100001} \\ &\quad\ \texttt{01110001 01001101 00000111} && \text{(bits)} \\ &= 231\ 15\ 133\ 142\ 178\ 224 && \text{(integers)} \end{aligned}$$

g_i^{nont}	n^{nont}	j^{nont}	g_i^{rule}	$\lvert R_s \rvert$	j^{rule}	Phenotype p
231	1	0	15	3	0	⟨**expr**⟩
133	3	1	142	4	2	(⟨expr⟩ ⟨**op**⟩ ⟨expr⟩)
178	2	0	224	3	2	(⟨**expr**⟩ * ⟨expr⟩)
231	2	1	15	3	0	(⟨var⟩ * ⟨**expr**⟩)
133	4	1	142	3	1	(⟨var⟩ * (⟨**expr**⟩ ⟨op⟩ ⟨expr⟩))
178	4	2	224	4	0	(⟨var⟩ * (⟨num⟩ ⟨**op**⟩ ⟨expr⟩))
231	3	0	15	2	1	(⟨**var**⟩ * (⟨num⟩ + ⟨expr⟩))
133	2	1	142	3	1	(y * (⟨num⟩ + ⟨**expr**⟩))
178	2	0	224	10	4	(y * (⟨**num**⟩ + ⟨num⟩))
231	1	0	15	10	5	(y * (4 + ⟨**num**⟩))
						(y * (4 + 5))

Fig. 3: Steps of the πGE mapping procedure with the genotype, grammar, and graphic convention of Figure 2.

shown experimentally that, for the majority of the problems, πGE outperformed standard GE [16], [30].

In details, in πGE each codon consists of a pair $g_i^{\text{nont}}, g_i^{\text{rule}}$ of integers, each of n bits, where n is set to 8 by convention. The mapping procedure is the same as GE, with the exception of step 1 where the non-terminal of p to be derived is the j^{nont}-th one (zero-based indexing), rather than the leftmost, with $j^{\text{nont}} = g_i^{\text{nont}} \mod n^{\text{nont}}$, n^{nont} being the number of non-terminals in p. Then, the derivation is performed using the j^{rule}-th production rule, with $j^{\text{rule}} = g_i^{\text{rule}} \mod |R_s|$.

Figure 3 shows an example of the mapping procedure of πGE.

C. SGE

Structured GE (SGE) [17] is one of the youngest variants of GE. In this framework, the linear genotype that characterizes standard GE and πGE is replaced by a structured one, where fixed-size lists of integers (*genes*) correspond to possible derivations of non-terminals. This representation ensures that the modification of a gene does not affect the derivation of other non-terminals, thereby increasing locality [31]. SGE has been showed to be more effective than standard GE [32] and also to exhibit better locality and lower degeneracy [31]. A more recent study showed that the interaction of genotype size, crossover, and diversity may reduce the degree to which SGE satisfies these properties [28].

SGE cannot be applied to recursive grammars, unlike GE, πGE, and WHGE. The reason for this limitation is in the fact that SGE lacks a mechanism for reusing the genotype. The inventors of SGE suggested a procedure for transforming any possibly recursive grammar $\mathcal{G}$ into a non-recursive grammar $\mathcal{G}'$ [17]: in order to use this procedure, the user must specify a maximum depth d_{max} for the derivation trees.

The genotype g in SGE is a fixed-length integer string which is composed of $|N|$ substrings (genes), that is, one substring g_s for each non-terminal $s \in N$ of the grammar $\mathcal{G}$. The length of each substring g_s is determined by the maximum number of derivations which can be applied to the corresponding non-terminal s according to the non-recursive grammar $\mathcal{G}'$; the domain of each codon in the gene is set to $\{0, \dots, |R_s| - 1\}$, R_s being the production rules for s. As pointed out in [17], by defining the genotype structure in this manner, SGE guarantees that the modification of a codon does not affect the derivation of other non-terminals, thus narrowing the number of changes that can occur at the phenotypic level.

$$g = \overbrace{0\,1\,0\,2\,2\,1\,1}^{\langle\text{expr}\rangle}\ \overbrace{0\,2\,3}^{\langle\text{op}\rangle}\ \overbrace{0\,1\,1\,0}^{\langle\text{var}\rangle}\ \overbrace{3\,7\,2\,4}^{\langle\text{num}\rangle}$$

$i_{\langle\text{expr}\rangle}$	$i_{\langle\text{op}\rangle}$	$i_{\langle\text{var}\rangle}$	$i_{\langle\text{num}\rangle}$	g_{s,i_s}	Phenotype p
0	0	0	0	0	⟨**expr**⟩
1	0	0	0	1	(⟨**expr**⟩ ⟨op⟩ ⟨expr⟩)
2	0	0	0	3	(⟨**num**⟩ ⟨op⟩ ⟨expr⟩)
2	0	0	1	0	(3 ⟨**op**⟩ ⟨expr⟩)
2	1	0	1	0	(3 + ⟨**expr**⟩)
3	1	0	1	2	(3 + (⟨**expr**⟩ ⟨op⟩ ⟨expr⟩))
4	1	0	1	0	(3 + (⟨**var**⟩ ⟨op⟩ ⟨expr⟩))
4	1	1	1	2	(3 + (x ⟨**op**⟩ ⟨expr⟩))
4	2	1	1	2	(3 + (x * ⟨**expr**⟩))
5	2	1	1	1	(3 + (x * ⟨**var**⟩))
					(3 + (x * y))

Fig. 4: Steps of the SGE mapping procedure with the grammar of Figure 1 and a genotype g of 18 integers (length determined upon the transformation of that grammar in a non-recursive grammar with $d_{\text{max}} = 4$). The rightmost column shows the phenotype p before the derivation of the highlighted non-terminal.

The mapping function of SGE is an iterative procedure in which, initially, the phenotype is set to $p = s_0$, and a counter i_s for each non-terminal $s \in N$ is set to 0—Figure 4 shows an example of execution. The following steps are then iterated.

1) The leftmost non-terminal s in p is derived by using the g_{s,i_s}-th production rule in R_s (zero-based indexing), with g_{s,i_s} denoting the value of the i_s-th codon (zero-based indexing) in g_s.
2) The counter i_s is incremented.

The procedure is iterated until no more non-terminals exist in p. It can be noted that SGE never aborts the mapping, hence it never gives invalid individuals.

While GE uses standard operators to explore the search space looking for good quality solutions, SGE uses tailored genetic operators able to work with the specific SGE representation. In particular, the mutation is reminiscent of the integer flip mutation also used in Genetic Algorithms. It consists in, for each codon, changing its value to a new random value in the appropriate domain, with a probability p_{mut}. Concerning crossover, it resembles the uniform crossover for bit string representation. It works by exchanging the genes g_s^1, g_s^2 of the parent genotypes corresponding to each non-terminal s in a randomly chosen subset $N' \subseteq N$.

III. WHGE

A. Overview

In this section, we provide an overview of our proposed mapping procedure. We describe the procedure in full detail in Section III-B and discuss the design rationale in Section III-C.

The mapping from the genotype into the phenotype occurs in two steps: the genotype is mapped to a derivation tree of the starting symbol of the CFG; the phenotype is then obtained by concatenating, from the left to the right, the leaf nodes of the derivation tree. The first mapping step is based on the following key ideas: (i) each node of the derivation tree is associated with a substring of the genotype; (ii) the genotype substring associated with a node is the concatenation of the substrings associated with the children of that node—hence the root node of the derivation tree is associated with the full genotype; and, (iii) the choice of the production rule for deriving a node depends *only* on the genotype substring associated with that node. This mapping introduces a form of *hierarchy* in the genotype.

Another important aspect of our contribution comes from the observation that different non-terminals of a grammar typically have widely differing *expressive power*, that is, they can result in many or few different sequences of terminals. For example, in the grammar of Figure 1, $\langle \mathsf{var} \rangle$ may be derived in 2 different mathematical expressions, $\langle \mathsf{num} \rangle$ in 10 different expressions, and $\langle \mathsf{expr} \rangle$ in, potentially, infinite different expressions. Associating the same number of bits with every child node, irrespective of its expressive power, would thus constitute an inefficient usage of the information encoded in the genotype. For this reason, WHGE does not split the genotype into pieces of equal length: WHGE associates each node with a number of bits that depend on the expressive power of that node. This feature corresponds to *weighting* each node in the hierarchy differently, which motivates the name that we have chosen for our proposal: Weighted Hierarchical mapping (WHGE).

We quantify the expressive power e_s of a symbol s with the number of different (partial) derivation trees with which can be generated from s ($e_s = 1$ for terminal symbols). We compute the expressive power for each symbol in advance, before starting the evolution, based on the specific grammar used. Since e_s could not be finite for non-terminals of recursive grammars (e.g., $\langle \mathsf{expr} \rangle$ in the grammar Figure 1), we count e_s only for derivation trees with a predefined maximum depth n_d, n_d being a parameter of WHGE: if a derivation tree still contains non-terminals at depth n_d, we do not further derive them, and count the resulting partial derivation trees without further deriving them. We remark, though, that the value of this parameter does not directly affect the maximum depth of phenotypes built with WHGE. The maximum depth of a phenotype is determined only by the grammar and the size $|g|$ of the genotype and is, in general, larger than n_d. In other words, unlike SGE, WHGE does not require to set the maximum depth of the phenotype in advance.

B. Mapping procedure

WHGE is based on a recursive function $\textsc{Map}(s, g')$ which takes as arguments a symbol $s \in N \cup T$ and a bit string g' and returns a derivation tree (this function is illustrated below). The mapping of a genotype g into a phenotype is obtained by calling $\textsc{Map}(s_0, g)$, s_0 being the starting symbol.

The function $\textsc{Map}(s, g')$ works as follows (Algorithm 1). If s is a terminal, the function returns a tree composed of the only symbol s. Otherwise, the following steps are performed (given a sequence or bit string x, we denote by $|x|$ the number of elements in the sequence of bits in the bit string, respectively):

1) Construct the sequence R_s of production rules for s ($\textsc{RulesFor}()$ in Algorithm 1).
2) Choose the i-th production rule in R_s, as follows.
   a) If $|g'| \geq |R_s|$, then:
      i) let $l_g := \left\lfloor \frac{|g'|}{|R_s|} \right\rfloor$; partition g' in $|R_s|$ non-overlapping substrings, as follows: the first $|g'| \bmod |R_s|$ substrings have length $l_g + 1$ and the remaining ones have length l_g ($\textsc{SplitForRule}()$);
      ii) set i equal to the index of the substring with largest *relative cardinality* (i.e., number of bits set to 1 divided by the length of the substring) ($\textsc{LargestCardIndex}()$); the handling of ties is explained below.
   b) Otherwise, if $|g'| < |R_s|$, set i equal to the index of the production rule in R_s which leads to a sequence of terminals in the lowest number of derivations from s ($\textsc{ShortestRuleIndex}()$)—the handling of ties is explained below.
3) Apply the production rule selected at the previous step to the input argument s of $\textsc{Map}()$ ($\textsc{ApplyRule}()$). Let $s_1, \dots, s_n$ be the n symbols resulting from that derivation.
4) If $n = 1$, remove the last bit from the input argument g' of $\textsc{Map}()$ ($\textsc{DropTrailingBit}()$). This step is required for preventing infinite recursion with certain recursive grammars, as explained below.
5) Partition g' in n non-overlapping substrings ($\textsc{SplitForChildren}()$) such that the length of the i-th substring g'_i is proportional to $\log_2(e_i)$, where e_i is the expressive power of symbol s_i. The details for distributing all bits of g' across the n substrings are given in Algorithm 2: function $\textsc{WeightedPartitioning}(|g'|, (e_1, \dots, e_n))$ is invoked by $\textsc{SplitForChildren}(g', (s_1, \dots, s_n))$ and returns the length of each substring g_i.
6) Build the tree t to be returned, by (recursively) invoking $\textsc{Map}()$ once for each of the symbols derived at step 3; each invocation takes as argument the symbol and the corresponding genotype portion selected at step 5; the trees returned by the invocations are appended as children of the node previously associated with the input argument s of $\textsc{Map}()$.

As an example of step 2b, assume $s = \langle \mathsf{expr} \rangle$ and consider the grammar of Figure 1. In this case, both the production rules $\langle \mathsf{expr} \rangle \to \langle \mathsf{var} \rangle$ and $\langle \mathsf{expr} \rangle \to \langle \mathsf{num} \rangle$ could be used,

Algorithm 1 WHGE genotype-phenotype mapping procedure. It is a recursive function initially invoked as $\textsc{Map}(s_0, g)$, s_0 being the starting symbol of the user-provided grammar.

```
function $\textsc{Map}(s, g')$
    $t \leftarrow \textsc{TreeNode}(s)$
    if $s \in N$ then                                    ▷ $s$ is a non-terminal
        $R_s \leftarrow \textsc{RulesFor}(s)$
        if $|g'| \geq |R_s|$ then                        ▷ $g'$ is long enough
            $(g''_1, \dots, g''_{|R_s|}) \leftarrow \textsc{SplitForRule}(g', |R_s|)$
            $i \leftarrow \textsc{LargestCardIndex}(g''_1, \dots, g''_{|R_s|})$
        else
            $i \leftarrow \textsc{ShortestRuleIndex}(R_s)$
        end if
        $(s_1, \dots, s_n) \leftarrow \textsc{ApplyRule}(R_s, i)$
        if $n = 1$ then
            $g' \leftarrow \textsc{DropTrailingBit}(g')$
        end if
        $(g'_1, \dots, g'_n) \leftarrow \textsc{SplitForChildren}(g', (s_1, \dots, s_n))$
        for $j \in \{1, \dots, n\}$ do
            $\textsc{AppendChild}(t, \textsc{Map}(s_j, g'_j))$
        end for
    end if
    return t
end function
```

Algorithm 2 Algorithm for the partitioning of a bit string based on its length and on the expressive power of symbols.

```
function $\textsc{WeightedPartitioning}(l, (e_1, \dots, e_n))$
    $e \leftarrow \sum_{i=1}^{i=n} \log_2 e_i$
    $(l_1, \dots, l_n) \leftarrow \left( \left\lfloor l \frac{\log_2 e_1}{e} \right\rfloor, \dots, \left\lfloor l \frac{\log_2 e_n}{e} \right\rfloor \right)$
    $c \leftarrow 0$
    while $l > \sum_{i=0}^{i=n} l_i$ do              ▷ distribute remaining bits, if any
        $j \leftarrow 1 + (c \mod n)$
        $l_j \leftarrow l_j + 1$
        $c \leftarrow c + 1$
    end while
    return $(l_1, \dots, l_n)$
end function
```

since they result in two derivations to a terminal, whereas the production rule $\langle \mathsf{expr} \rangle \rightarrow$ ($\langle \mathsf{expr} \rangle$ $\langle \mathsf{op} \rangle$ $\langle \mathsf{expr} \rangle$) would require at least three derivations. Note that the implementation of $\textsc{ShortestRuleIndex}()$ at step 2b can rely on data computed in advance, before starting the evolution, as it suffices to analyze each non-terminal based only on the grammar.

Ties at step 2 are handled as follows. Let n_{ties} denote the number of ties, i.e., substrings of g' with maximal relative cardinality ($\textsc{LargestCardIndex}()$) or production rules which lead to a sequence of terminals with minimal number of derivations ($\textsc{ShortestRuleIndex}()$). In both cases we construct a list with all the n_{ties} candidate items and select the item whose position in the list is equal to the remainder of the division between $|g'|$ and n_{ties}; in case g' is empty, which might occur with recursive grammars as explained in the next paragraph, we use the length of the full genotype $|g|$ as dividend. The motivation for this choice is to avoid the introduction of any bias in the mapping procedure, which could itself make some regions of the phenotype space harder to be explored.

Step 4 prevents infinite recursion with certain recursive grammars, as explained below. When execution of step 3 results in $n = 1$, the result of $\textsc{SplitForRule}(g', n)$ consists of one single element identical to the full input argument g' of $\textsc{Map}()$. In the absence of the last bit removal ($\textsc{DropTrailingBit}()$), the subsequent recursive invocation of $\textsc{Map}()$ would take again g' as argument. With certain recursive grammars, this flow could result in infinite recursion. For example, consider the call of $\textsc{Map}(\langle \mathsf{a} \rangle, \mathsf{1110})$, i.e., $s = \langle \mathsf{a} \rangle$ and $g' = \mathsf{1110}$, with a grammar such that the production rules $R_{\langle \mathsf{a} \rangle}$ for s are given by $\langle \mathsf{a} \rangle$ ::= $\langle \mathsf{a} \rangle$ | b: step 2(a)ii would cause the selection of the first production rule (since $\|\mathsf{11}\| > \|\mathsf{10}\|$), which would result in splitting g' in $n = 1$ portion (i.e., g' itself), with $s_1 = \langle \mathsf{a} \rangle$, eventually leading to calling again $\textsc{Map}(\langle \mathsf{a} \rangle, \mathsf{1110})$. By removing one bit at Step 4 we instead ensure that the second argument of $\textsc{Map}()$ (i.e., g') becomes shorter upon each invocation and eventually becomes the empty bit string. Therefore, the condition $|g'| \geq |R_s|$ (step 3) will eventually switch from true to false and the selected production rule will eventually change.

Figure 5 shows an example of the mapping procedure of WHGE with $n_d = 2$.

C. Design discussion

One of the key motivations for our proposal was the observation that imposing a structure on the genotype may have highly beneficial effects over approaches based on a purely linear genotype, as advocated and demonstrated by SGE. Differently from SGE, however, we aimed at designing a mapping that could fit the overall GE framework without requiring any dedicated handling of user-provided grammars or specialized genetic operators. In this respect, we believe that hierarchical relations between nodes of a derivation tree are the most natural way for imposing a structure on the genotype. We were encouraged to tailor such structure by weighting grammar symbols differently, based on the results of early experiments [29]. The intuition that drove this choice was that varying the size of each genotype portion depending on the expressive power of the derived symbol is a way for encoding information in the genotype more efficiently.

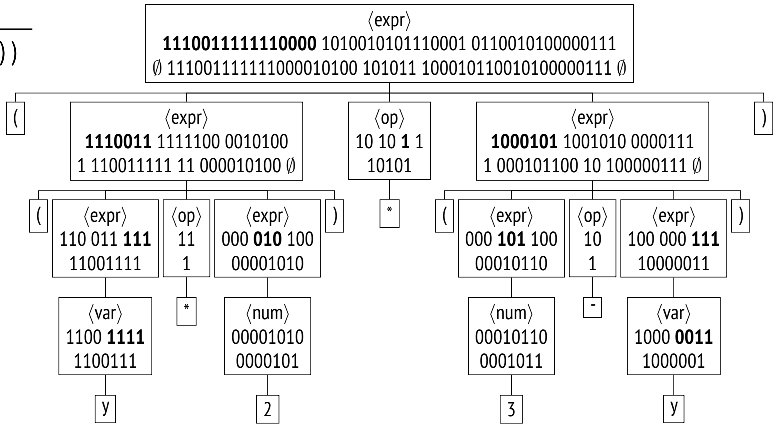
While designing the details of the WHGE mapping we followed design principles aimed at obtaining better invalidity, degeneracy, and locality properties than those of the standard GE mapping (see below). We note, though, that whether better mapping properties may indeed lead to a more effective search is an open research question [33], [34]. We also emphasize that we cannot prove that WHGE is guaranteed to exhibit better properties than GE, with every possible grammar. We assess the resulting properties of WHGE experimentally, on a broad range of benchmarks (Section IV-C).

Existing literature indicates that invalidity (the tendency of generating non-valid individuals) does not provide any

$g =$ 111001111111000010100101011100010110010100000111

Args.		Inner values			Return val.
s	$\lvert g'\rvert$	$\lvert R_s\rvert$	i	$(\lvert g'_1\rvert, \dots, \lvert g'_n\rvert)$	t
⟨expr⟩	48	3	0	$(0, 21, 6, 21, 0)$	((y*2)*(3-y))
⟨expr⟩	21	3	0	$(1, 9, 2, 9, 0)$	(y*2)
⟨expr⟩	9	3	2	(8)	y
⟨var⟩	8	2	1	(7)	y
⟨op⟩	2	4	2	(1)	*
⟨expr⟩	9	3	1	(8)	2
⟨num⟩	8	10	2	(7)	2
⟨op⟩	6	4	2	(5)	*
⟨expr⟩	21	3	0	$(1, 9, 2, 9, 0)$	(3-y)
⟨expr⟩	9	3	1	(8)	3
⟨num⟩	8	10	3	(7)	3
⟨op⟩	2	4	1	(1)	-
⟨expr⟩	9	3	2	(8)	y
⟨var⟩	8	2	1	(7)	y

(a) MAP() invocations during the mapping procedure.



(b) Decorated derivation tree.

Fig. 5: Detailed description of the WHGE mapping of an example genotype g with the grammar of Figure 1 and maximum tree depth $n_d = 2$. In that grammar, the expressive power of non-terminal symbols ⟨expr⟩, ⟨op⟩, ⟨var⟩, and ⟨num⟩ are 66, 4, 2, and 10, respectively. The left figure contains one row for each recursive invocation of $\textsc{Map}(s, g')$ (in depth-first order), described with arguments, internal values, and return value (as concatenation of leaf nodes). The return value of the first row is thus the phenotype resulting from the mapping. Indentation levels emphasize the recursion depth. The right figure shows the corresponding decorated derivation tree, each node associated with an invocation to $\textsc{Map}(s, g')$. Each node contains symbol s, the genotype g', and the portions of g' that will be passed at the next recursive invocations (one for each node child). The genotype g' is represented as split by SPLITFORRULE(), with the portion chosen by LARGESTCARDINDEX() for selecting the grammar rule highlighted in bold; g' is not split when the grammar rule is selected by SHORTESTRULEINDEX() ($\emptyset$ denotes the zero-length bit string).

beneficial effect and is detrimental to the evolutionary process [28], [27]. Representations with this property are used either by associating non-valid individuals with the worst possible fitness [1] or by discarding them and generating new ones [35], [36]—the latter resulting in wasting computational resources. Based on these considerations, one of the basic design principles of WHGE is that non-valid individuals should not exist: indeed, in WHGE, every genotype may always be mapped into a phenotype. In this respect, it can be noted that WHGE never aborts the mapping, differently than GE. This guarantee directly derives from the WHGE mapping procedure since: (a) at each derivation, the size of the genotype substring associated with the resulting non-terminals is strictly lower than the size of the genotype substring of the derived non-terminal; and (b) when the genotype substring associated with a non-terminal to be derived is too short, a predefined production rule is chosen which will eventually lead to a sequence composed of only terminals. These two conditions ensure that endless executions of the mapping procedure cannot occur, hence eventually delivering a phenotype.

Degeneracy (the tendency of mapping multiple different genotypes on the same phenotype [37], [23]) is one of the most prominent properties of the representation [20]. Some studies speculated that degeneracy may be beneficial to the search effectiveness, on the grounds that a highly degenerated representation might over-represent the optimal solution, hence increasing the likelihood of a fast convergence towards that solution [21]. More specific arguments along this line were provided in [38]: the authors claimed that (i) degeneracy is responsible for the preservation of the functionality of the phenotype, while still allowing an unrestricted search of the genotypic search space, and (ii) degeneracy in the genetic code has a beneficial effect on the genotypic diversity of the population. However, we designed WHGE by considering that degeneracy is an *undesirable* property, based on several recent studies that point in this direction [28], [39], [27], [31]. Significant arguments in this respect are that a representation with high degeneracy tends to over-represent those phenotypes which are too simple to be effective [27], and that degeneracy tends to be inversely correlated with evolvability, which might be explained on the grounds that the tendency of changing genotypes without changing the resulting phenotypes is detrimental to the chances of improving fitness [39].

We sought to minimize degeneracy by attempting to minimize a related but different property, i.e., redundancy (the tendency of not using portions of the genotype for mapping into the phenotype) [37], [23]. Redundancy is one of the sources for degeneracy since differences in the unused portions of the genotype cannot result in different phenotypes [25], hence the greater the redundancy, the greater the degeneracy. For representations based on bit strings, redundancy may be visualized with the DU maps introduced in [40]. In WHGE we attempt to minimize redundancy by ensuring that every mapping execution analyzes all bits of the genotype, unlike GE in which there may be mapping executions that complete without using all the bits in a genotype. Of course, using all bits of the genotype does not necessarily imply that all of them play a crucial role in determining the phenotype. Indeed, the WHGE mapping does not prevent degeneracy: for example, there might be different genotypes that lead to choosing the same value for index i at step 2(a)ii, even though the content of the i-th substring is different for each genotype.

Locality (the tendency of mapping genotypic neighbors to phenotypic neighbors) is another property of a representation that has received much attention recently [28], [27]. Existing literature suggests that locality is beneficial for the quality of the evolutionary process as a whole [26], [22], but it has recently been shown that high degeneracy could nullify the potential advantages of high locality [39]. In other words, the potential advantages of a representation with high locality depend also on the other properties of the representation. We will analyze this issue in depth in Section IV. In WHGE we attempted to improve locality by defining a mapping procedure in which the choice of the production rule tends to be more robust w.r.t. small modifications in the genotype than with standard GE, in particular, for nodes that are close to the root of the derivation tree. In this respect, consider a difference of a single bit in the initial portion of a genotype. In GE the production rule is chosen according to the remainder of a division, thus that single bit will modify the choice of the first production rule. It follows that all subsequent derivations will likely be modified as well, thereby resulting in a very different phenotype. In WHGE the production rule for the root node and for nodes close to the root is chosen with a criterion that is likely unaffected by the swapping of a single bit, that is, the relative cardinality on genotype substrings.

IV. Experimental evaluation

A. Benchmark problems

We performed a number of experiments in order to thoroughly assess our WHGE proposal. We used a set of 9 benchmark problems which we chose considering the guidelines for the evaluation of Genetic Programming approaches proposed in [18], [19]. In particular, we considered 2 Boolean, 3 synthetic, and 4 symbolic regression problems (among which 3 include a testing set different from the training set used during the evolution), which follow:

- MOPM-3: Multiple outputs parallel 3-bit multiplier—the value of 3 being chosen as a reasonable intermediate value w.r.t. the value of 5, which has been shown to be the largest for which a correct solution has been evolved [41]. The fitness is given by the number of errors among all the input cases.
- Parity-5: 5-bit parity. We included this benchmark, despite being considered by some rather trivial, because GE and πGE struggle in evolving an effective solution. The fitness is given by the number of errors among all the input cases.
- KLandscapes-3 and KLandscapes-7: K Landscapes with $k = 3$ and $k = 7$, a tunable, GP-specific benchmark which has been proposed recently [42]. We built a simple CFG for expressing the corresponding trees; moreover, we here express the fitness of a solution t as $f(t) = 1 - f_0(t)$, where $f_0(t)$ is the original fitness function described in [42], in order to make it consistent with the other problems, for which the lower the fitness, the better.
- Text [28]: generation of a target string `Hello world!`; the fitness is given by the edit distance between the string corresponding to the solution and the target string. The grammar of Text is more complex (see Figure 6) than those of the other benchmark problems, both in the depth of the dependencies among non-terminals and in the number of production rules for each non-terminal.
- Keijzer6 [43]: symbolic regression of the function $f(x) = \sum_{i=1}^{x} \frac{1}{i}$, with a training set of 50 points evenly spaced in $[1, 50]$ and a testing set of 50 points evenly spaced in $[1, 120]$.
- Nguyen7 [44]: symbolic regression of the function $f(x) = \log{(x+1)} + \log{(x^2+1)}$, with a training set of 20 points uniformly sampled in $[0, 2]$.
- Pagie1 [45]: symbolic regression of the function $f(x, y) = \frac{1}{1+x^{-4}} + \frac{1}{1+y^{-4}}$, with a training set of 125 points resulting from 25 values evenly spaced in $[-5, 5]$ for both x and y and a testing set (as done in [31]) of $10\,000$ points resulting from 100 values evenly spaced in the same interval for both x and y.
- Vladislavleva4 [46]: symbolic regression of the function $f(x_1, \dots, x_5) = \frac{10}{5+\sum_{i=1}^{5}(x_i-3)^2}$, with a training set of 1024 points uniformly sampled in $[0.05, 6.25]^5$ and a testing set of 5000 points uniformly sampled in $[-0.25, 6.35]^5$.

For all the symbolic regression problems, the fitness is given by the sum of the absolute errors between target and obtained values. Figure 6 shows the CFGs for the 9 benchmark problems.

B. Procedure and baselines

We performed 30 runs, by varying the random seed, for each of the 4 variants (the original GE, πGE, SGE, and WHGE) and each of the 9 benchmark problems. We used the evolutionary parameters shown in Table I.

Concerning the variant-specific parameters, we set: the genotype size to 1024 bits for GE, πGE, and WHGE; the maximum number of wrappings to $n_w = 1$ for GE and πGE; the maximum tree depth to $d_{\text{max}} = 6$ for SGE (as suggested by its inventors); and the maximum depth for determining the expressive power of non-terminals to $n_d = 3$ for WHGE (Section III-A).

We performed the experimentation with an evolutionary framework for grammar-based GP which we developed in

⟨o⟩ ::= ⟨e⟩ ⟨e⟩ ⟨e⟩ ⟨e⟩ ⟨e⟩ ⟨e⟩
⟨e⟩ ::= .or ⟨e⟩ ⟨e⟩ | .xor ⟨e⟩ ⟨e⟩ | .and ⟨e⟩ ⟨e⟩ |
  .and1not ⟨e⟩ ⟨e⟩ | ⟨v⟩
⟨v⟩ ::= v1.1 | v1.2 | v1.3 | v2.1 | v2.2 | v2.3

(a) MOPM-3

⟨e⟩ ::= .or ⟨e⟩ ⟨e⟩ | .and ⟨e⟩ ⟨e⟩ | .not ⟨e⟩ | ⟨v⟩
⟨v⟩ ::= v1 | v2 | v3 | v4 | v5

(b) Parity-5

⟨N⟩ ::= ⟨n⟩ ⟨N⟩ ⟨N⟩ | ⟨t⟩
⟨n⟩ ::= n0 | n1
⟨t⟩ ::= t0 | t1 | t2 | t3

(c) KLandscapes-3 and KLandscapes-7

⟨text⟩ ::= ⟨sentence⟩ ⟨text⟩ | ⟨sentence⟩
⟨sentence⟩ ::= ⟨Word⟩ ␣ ⟨sentence⟩ | ⟨word⟩ ␣ ⟨sentence⟩ |
  ⟨word⟩ ⟨punct⟩
⟨word⟩ ::= ⟨letter⟩ ⟨word⟩ | ⟨letter⟩
⟨Word⟩ ::= ⟨Letter⟩ ⟨word⟩
⟨letter⟩ ::= ⟨vowel⟩ | ⟨consonant⟩
⟨vowel⟩ ::= a | e | i | o | u
⟨consonant⟩ ::= b | c | d | ... | z
⟨Letter⟩ ::= ⟨Vowel⟩ | ⟨Consonant⟩
⟨Vowel⟩ ::= A | E | I | O | U
⟨Consonant⟩ ::= B | C | D | ... | Z
⟨punct⟩ ::= ! | ? | .

(d) Text

⟨expr⟩ ::= ⟨op⟩ ⟨expr⟩ ⟨expr⟩ | pre-op ⟨expr⟩ | ⟨var⟩
⟨op⟩ ::= + | *
⟨pre-op⟩ ::= uminus | 1/ | sqrt
⟨var⟩ ::= x

(e) Keijzer6

⟨expr⟩ ::= ⟨op⟩ ⟨expr⟩ ⟨expr⟩ | pre-op ⟨expr⟩ | ⟨var⟩
⟨op⟩ ::= + | - | p/ | *
⟨pre-op⟩ ::= sin | cos | exp | plog
⟨var⟩ ::= x | 1.0

(f) Nguyen7

⟨expr⟩ ::= ⟨op⟩ ⟨expr⟩ ⟨expr⟩ | pre-op ⟨expr⟩ | ⟨var⟩
⟨op⟩ ::= + | - | p/ | *
⟨pre-op⟩ ::= sin | cos | exp | plog
⟨var⟩ ::= x | y | 1.0

(g) Pagie1

⟨expr⟩ ::= ⟨op⟩ ⟨expr⟩ ⟨expr⟩ | pre-op ⟨expr⟩ | ⟨var⟩ | ⟨const⟩
⟨op⟩ ::= + | - | p/ | *
⟨pre-op⟩ ::= squared
⟨var⟩ ::= x1 | x2 | x3 | x4 | x5
⟨const⟩ ::= 1.0 | 2.0 | 3.0 | ... | 10.0

(h) Vladislavleva4

Fig. 6: The grammars of the benchmark problems: in the symbolic regression problems, p/ and plog are the protected versions of the division and the logarithm, respectively.

TABLE I: Parameters for the evolutionary runs.

Population	500
Pop. initialization	Random
Generations	50
Crossover op.	two-points same (GE, πGE, WHGE) SGE crossover (SGE)
Crossover rate	0.8
Mutation op.	bit flip w. $p_{\text{mut}} = 0.01$ (GE, πGE, WHGE) SGE mutation w. $p_{\text{mut}} = 0.01$ (SGE)
Selection	tournament with size 3
Replacement	$m + n$ strategy, w. $m = n$ and overlapping

Java. The framework implements all the 4 variants and the 9 benchmark problems and is publicly available on Github[1].

C. Results and discussion

In this section, we compare the fitness values achieved by the original GE, πGE, SGE, and WHGE from several points of view.

Table II presents the median of the fitness of the best individuals at the end of the evolution, across the 30 repetitions, for all the different problems and variants.

It can be seen that WHGE obtains the best median fitness in 6 of the 9 considered benchmarks—strictly better than the other approaches in 4 of them—and the second best median fitness in each of the 3 remaining benchmarks. When the WHGE is the second best performer, the difference between the best performer is minimal in 2 of the 3 benchmarks: 2 vs. 0 for Parity-5 (16 for the third performer) and 5.1 vs. 5.0 for Keijzer6 (5.8 for the third performer). These results are a strong indication, we believe, of the potential of the proposed WHGE mapping. Indeed, we found similar results in terms of fitness improvement with WHGE even with genotype size shorter than 1024 bits, i.e., with 128 bits [47] and with 256 bits [48] (we refer the reader to the cited works for full details).

To assess the statistical significance of the results obtained, we performed a set of tests. Initially, we applied the Lilliefors test to verify if the data comes from a normal distribution, against the alternative that it does not come from such a distribution. The result of the test, performed with a significance level of 5%, suggested that the alternative hypothesis cannot be rejected. Then, we considered a rank-based statistics and performed the Mann-Whitney U-test to verify if the samples have equal medians, against the alternative that they have not. In this case the test indicated that the difference in terms of fitness between the proposed WHGE mapping and GE, πGE and SGE is indeed statistically significant in several of the benchmarks taken into account (Table III, we used a value of $\alpha = 0.05$ with a Bonferroni correction in both the tests, Mann-Whitney and Lilliefors). This is a further corroboration of the findings in Table II.

Fitness values presented in Table II are graphically represented in the box plots of Figure 7. On each box, the central

[1] https://github.com/ericmedvet/evolved-ge

TABLE II: Best fitness upon the last generation, median value (Q_2) and standard deviation (σ) across the 30 runs. For each problem, the best median value among GE variants is highlighted in bold.

	Boolean				Other						Symbolic regression							
	MOPM-3		Parity-5		KLand.-3		KLand.-7		Text		Keijzer6		Nguyen7		Pagie1		Vlad.4	
Method	Q_2	σ	Q_2	σ	Q_2	σ	Q_2	σ	Q_2	σ	Q_2	σ	Q_2	σ	Q_2	σ	Q_2	σ
GE	53	6.7	16	6.9	0.38	0.13	0.82	0.04	**5.0**	1.3	**5.0**	2.5	**0.33**	0.24	2.3	1.56	141	5.8
πGE	65	6.1	16	5.7	0.38	0.13	0.92	0.02	**5.0**	1.3	5.8	1.6	**0.33**	0.39	3.2	1.94	141	2.6
SGE	42	2.9	**0**	1.9	**0.19**	0.06	0.98	0.00	6.0	0.5	6.8	2.5	0.58	0.07	3.5	0.97	141	2.0
WHGE	**38**	3.2	2	2.2	0.38	0.20	**0.59**	0.03	**5.0**	1.1	5.1	0.9	**0.33**	0.11	**1.8**	0.74	**138**	2.5

TABLE III: p-values returned by the Mann-Whitney U-Test on the Best Fitness. Bold denotes statistically significant values.

Problem		πGE	SGE	WHGE
MOPM-3	GE	**0.005**	**0**	**0**
	πGE		**0**	**0**
	SGE			**0**
Parity-5	GE	0.819	**0**	**0**
	πGE		**0**	**0**
	SGE			0.01
KLand.-3	GE	0.021	**0**	**0**
	πGE		**0**	0.031
	SGE			**0**
KLand.-7	GE	0.021	**0**	**0**
	πGE		**0**	0.031
	SGE			**0**
Text	GE	0.933	**0**	0.517
	πGE		**0**	0.464
	SGE			**0**
Keijzer6	GE	0.011	**0**	0.807
	πGE		**0**	**0.001**
	SGE			**0**
Nguyen7	GE	0.865	**0**	0.487
	πGE		**0.005**	0.549
	SGE			**0**
Pagie1	GE	0.032	0.019	0.064
	πGE		0.853	**0**
	SGE			**0**
Vlad.4	GE	0.347	0.300	0.232
	πGE		0.090	0.028
	SGE			0.154

mark is the median, the edges of the box are the 25th and 75th percentiles, and the whiskers extend to the most extreme data points not considered outliers. For the Keijzer6, Pagie1, and Vladislavleva4 problems, there are two groups of boxes, one associated with the input cases (as for the other problems) and another in which the best individual is assessed on a separated testing set (these benchmarks are the only ones for which such a testing set is available).

Further insights into the fitness values may be obtained from Figure 8, which shows the median fitness value of the best individual during the evolution. While these graphs do not allow drawing any general conclusions, it can be observed that in most problems the initial values for WHGE tend to be better than with GE/πGE. This finding could be explained by the lower degeneracy of WHGE (see Section IV-D) which results in a tendency of WHGE to better sample the phenotype space given a random set of genotypes. In other words, WHGE might be more robust to the population initialization procedure—a key step in GE [49]—than GE/πGE.

Finally, we provide in Figure 9 a scatter plot showing the fitness and the phenotype length of the best solutions obtained in all the 30 runs that we executed. It can be seen that WHGE tends to produce larger phenotypes than those of the other variants. This fact could explain, in particular, the much better fitness exhibited in MOPM-3, Parity-5, and KLandscapes-7 with respect to the other competitors. In the case of KLandscapes-3, on the other hand, longer phenotypes do not result in better fitness. We believe this result is related to the fact that, in this specific benchmark, trees with a depth larger than 3 are penalized in terms of fitness. Since WHGE is not biased towards small phenotypes, unlike other GE variants [27], a WHGE search will start from a region of the search space which is, in general, farther from the optimal solution. Thus, for KLandscapes-3, the ability of WHGE to indeed generate fitter individuals during the search (see the evolvability analysis in the next section) turns out to be not sufficient to obtain good solutions. It is also interesting to observe that SGE tends to produce much shorter phenotypes that WHGE and with much smaller variance, in all the considered benchmarks. The ability of WHGE to construct longer phenotypes could be crucial for solving, e.g., KLandscapes-7, more efficiently. This fact deserves further investigation, though.

D. Mapping properties

In this section, we analyze the mapping properties of WHGE and of the other variants. We consider invalidity, degeneracy, locality, evolvability (defined in Section I and recalled in Section III-C), and neutrality (defined below). To the best of our knowledge, this is the first comparative assessment of all these properties for several GE variants.

For each of the 4 mapping variants and each of the 9 benchmark problems: (i) we randomly generated a set G of $10\,000$ genotypes; (ii) we mapped each element of G into a phenotype. We then measured *invalidity* as $1 - \frac{|G_V|}{|G|}$ and *degeneracy*, as $1 - \frac{|P|}{|G_V|}$, where P is the set of phenotypes and G_V is the subset of G containing the elements for which the mapping did not abort.

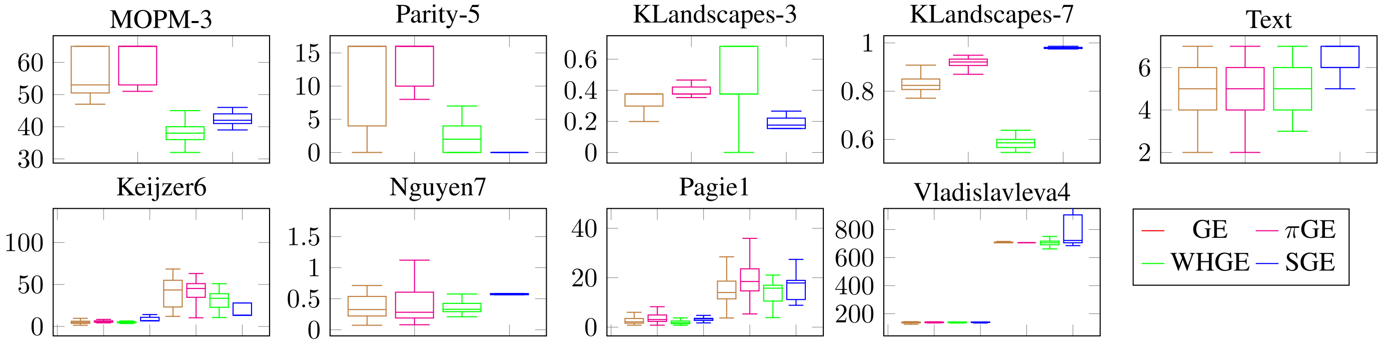


Fig. 7: Box plots of the best individual upon the last generation. For the Keijzer6, Pagie1, and Vladislavleva-4 problems, there is an additional group of boxes that describe the assessment of the best individual on a separated testing set.

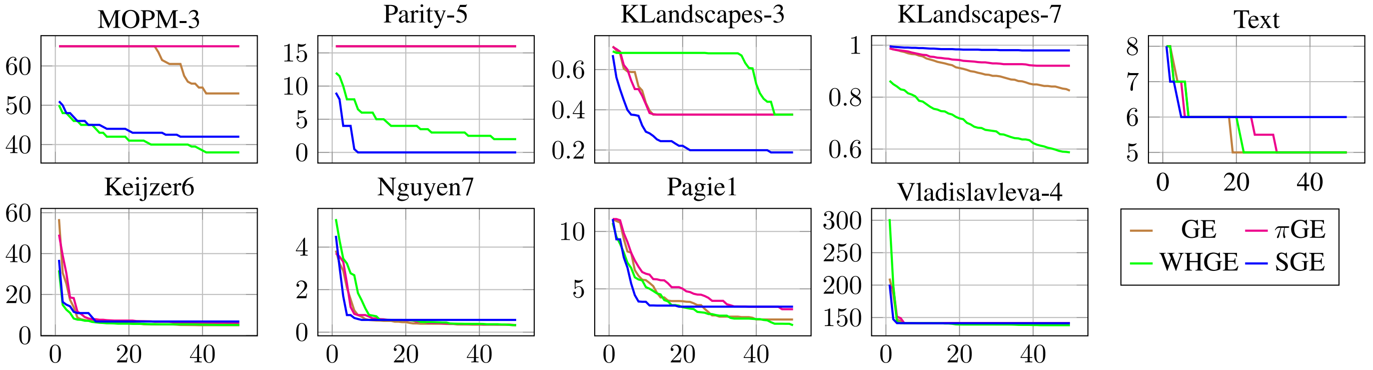


Fig. 8: Best fitness during the evolution, median value across the 30 runs.

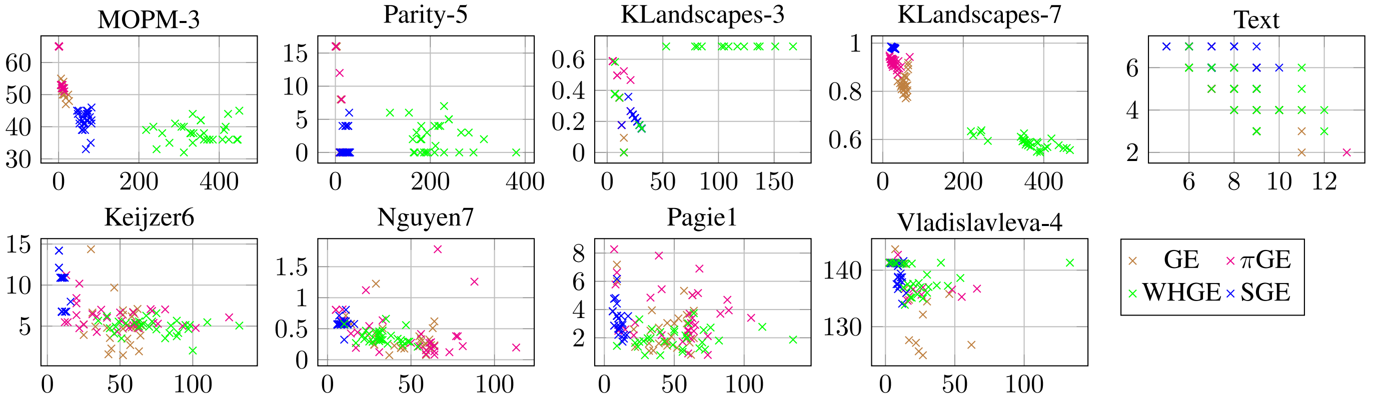


Fig. 9: Fitness vs. phenotype length of the best individual at the end of the evolution.

Concerning *locality*, (i) we selected a subset of 10 000 *pairs* of genotypes randomly chosen among the 10^8 pairs of G^2 and determined the corresponding pairs of phenotypes; (ii) we computed the genotype (Hamming distance) and phenotype (tree edit distance with the algorithm of [50]) distances between corresponding elements of each pair; (iii) we measured locality, as the Pearson correlation of genotype and phenotype distance in the same pair (that is the same approach as [28], [39]). We chose to use this locality measure, instead of one based on the (re-iterated) application of the mutation operator (as in, e.g., [31], [22]), because we deal with different representations based on different mutation operators.

We remark that the above procedure measures invalidity, degeneracy, and locality in a *static* context, because we attempted to exclude any factors related to the evolution dynamics from the analysis, e.g., lack of diversity in advanced stages of the evolution [28].

The analysis of evolvability (the tendency of generating fitter individuals during the search) and of neutrality (the combined tendency of a genetic operator and a mapping procedure to lead to the same phenotype) [25] instead requires a *dynamic* context. To this end, we instrumented the evolutionary framework used for the experiments in order to log, after each genetic operator application, the genotypes, phenotypes, and

TABLE IV: Static properties of the mapping: invalidity, degeneracy, and locality. For each problem, the lowest (all but locality) or greatest (locality) value is highlighted in bold. NaN means that the Pearson correlation cannot be computed, as the standard deviation of the distances among phenotype is equal to 0.

		Boolean		Other			Symbolic regression			
	Method	MOPM-3	Parity-5	KLand.-3	KLand.-7	Text	Keijzer6	Nguyen7	Pagie1	Vlad.4
Invalidity	GE	1	0.75	0.1	0.1	0.02	0.13	0.13	0.13	0
	πGE	1	0.76	0.13	0.13	0.08	0.16	0.16	0.16	0.01
	SGE	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	WHGE	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Degener.	GE	1	0.97	0.75	0.75	**0.11**	0.72	0.66	0.65	0.68
	πGE	1	0.97	0.77	0.77	0.17	0.75	0.69	0.68	0.68
	SGE	**0**	**0.22**	0.65	0.65	0.25	0.63	0.54	0.52	0.68
	WHGE	**0**	0.23	**0.63**	**0.63**	0.2	**0.42**	**0.41**	**0.41**	**0.61**
Locality	GE	−0.02	0	−0.01	−0.01	0.03	0.01	0	0.02	0
	πGE	NaN	−0.01	0.02	0.02	0.01	0.01	−0.01	0	0.02
	SGE	**0.08**	**0.08**	**0.14**	**0.14**	**0.33**	**0.15**	**0.07**	0.03	**0.09**
	WHGE	0.06	0.02	0	0	0.02	−0.01	0.03	**0.03**	0.02

TABLE V: Average neutrality and AEP (values ×100) during the evolution, median value across the 30 runs for each of the two genetic operators. For each problem, the lowest (neutrality) or greatest (AEP) median value is highlighted in bold.

		Boolean		Other			Symbolic regression			
	Method	MOPM-3	Parity-5	KLand.-3	KLand.-7	Text	Keijzer6	Nguyen7	Pagie1	Vlad.4
AEP mut.	GE	0	0	1.14	7.8	1.35	2.25	1.86	2.88	2.16
	πGE	0	0.09	1.84	2.98	1.75	2.01	1.94	2.77	2.5
	SGE	0.77	0.27	0.84	1.99	0	0.28	0.41	0.47	0.47
	WHGE	**1.27**	**0.79**	**3.53**	**12.02**	**2.48**	**6.69**	**7.31**	**7.88**	**4.98**
AEP cr.	GE	0	0	0.66	3.89	0.62	1.58	1.42	2.3	0.52
	πGE	0	0	0.91	3.28	0.82	3.42	3.57	4.55	1.51
	SGE	0.56	0.5	**8.71**	**15.09**	0.32	2.97	3.29	4.01	**5.46**
	WHGE	**0.75**	**0.56**	2.89	9.86	**2.11**	**5.37**	**6.89**	**7.76**	4.76
Neut. mut.	GE	0.42	0.87	0.75	0.44	0.42	0.14	0.17	0.14	0.71
	πGE	0.35	0.77	0.51	0.14	0.18	0.1	0.11	0.1	0.45
	SGE	0.24	0.13	0.13	0.13	0.12	0.16	0.29	0.32	0.46
	WHGE	**0**	**0**	**0.05**	**0.02**	**0.06**	**0.06**	**0.05**	**0.04**	**0.18**
Neut. cr.	GE	0.94	0.97	0.72	0.36	0.67	0.44	0.46	0.41	0.83
	πGE	0.86	0.93	0.64	0.3	0.42	0.27	0.25	0.24	0.63
	SGE	0.01	**0**	**0.01**	**0.01**	0.18	0.14	0.17	0.08	0.14
	WHGE	**0**	0.01	0.03	0.02	**0.04**	**0.03**	**0.02**	**0.02**	**0.07**

fitness values of the parents and the children. We then executed the same experiments described in Section IV-B.

Based on the accumulated events, we measured *evolvability* with the Accumulated Escape Probability (AEP) index [39], which essentially represents the average probability of obtaining a child which is fitter than its parents. We measured *neutrality* separately for each of the two genetic operators used. For mutation, we measured neutrality as the ratio of births in which the child phenotype is equal to the parent genotype and the two genotypes are different. For crossover, we measured neutrality as the ratio of births in which the child phenotype is equal to at least one parent phenotype and its genotype is different from both parent genotypes.

Both evolvability and neutrality are relevant properties for the different evolutionary algorithms, and they are not just peculiar to GE frameworks. Several different ways for quantifying these properties have been proposed, in order to capture the different nuances of the neutrality or for adapting the measure to the particular EA considered: e.g., [51], [52] for evolvability and [53], [54], [23], [55] for neutrality. We chose to measure evolvability with the method introduced in [56] and later used in [39] for GE: while in [56] evolvability is used to compare different problems tackled with the same representation, in [39] the same measure is used to compare different representations on the same set of problems, as in the present work.

Table IV shows the results for the static context (invalidity, degeneracy, locality), whereas Table V shows those for the dynamic context (evolvability and neutrality, separately for the two genetic operators).

The foremost finding from Table IV is that degeneracy in WHGE is in general much lower than in GE/πGE; the difference is smaller w.r.t. SGE, but still evident. Our attempt to involve all genotype bits in determining the resulting phenotype (Section III-C), thus, has succeeded, at least on the considered benchmarks. While WHGE does improve over GE and πGE in terms of locality, SGE has even better locality.

It is perhaps more interesting to observe that WHGE and SGE exhibit a sort of specular behavior in terms of degeneracy and locality: WHGE tends to exhibit the best degeneracy among all the variants while SGE tends to exhibit the best locality. Furthermore, the only benchmark in which SGE has better degeneracy than WHGE (Parity-5) is also one of the two benchmarks in which SGE manages to deliver a fitness better than WHGE (Table II), the difference being negligible in both the fitness and the degeneracy. In our experimental setting, thus, degeneracy seems to be more correlated with solution effectiveness than locality. As we have observed in Section III-C, though, a principled framework for using mapping properties as a proxy for predicting, or justifying, solution effectiveness is still lacking [33], [34].

Another interesting finding concerns the invalidity, an undesirable property which is, by design, equal to 0 in WHGE, as well as in SGE. We remark, however, that null invalidity is obtained in SGE by requiring the user to set a maximum depth for the derivations while mapping the genotype into phenotypes; in our WHGE, this requirement is not present.

Table V indicates clearly that WHGE exhibits a better evolvability than the other variants. The only exception is for the crossover operator in 3 benchmarks, in which WHGE is second-ranked and SGE is the first-ranked. The improvement over both GE and πGE is significant in all benchmarks. Indeed, our observations of high evolvability, low degeneracy, and good solution quality is consistent with the findings of [39]. It is interesting to observe the good evolvability of SGE for the crossover operator. We interpret this result as a consequence of the coupling between the structure of SGE genotype and peculiarities of SGE crossover. Table V also indicates that WHGE exhibits a better neutrality than the other variants, again with the exception of 3 benchmarks for the crossover operator in which SGE is the first-ranked and WHGE is second-ranked. The improvement over both GE and πGE is significant in all cases.

V. Concluding remarks

Imposing a structure on the genotype may have highly beneficial effects over the genotype-phenotype mapping in GE, as advocated and demonstrated by the recent proposal SGE. In this work, we have proposed a novel mapping for GE which imposes a form of hierarchy on the genotype and encodes grammar symbols with a varying number of bits based on the relative expressive power of those symbols. The proposed weighted hierarchical (WHGE) mapping does not impose any constraint on the overall GE framework, in particular, WHGE may handle recursive grammars, uses the classical genetic operators and does not need to define any bound in advance on the size of phenotypes.

We assessed experimentally our proposal in depth considering a set of benchmarks selected based on the guidelines for the evaluation of Genetic Programming approaches. Our results showed that WHGE obtains the best median fitness in 6 of the 9 considered benchmarks (strictly better than the other approaches in 4 of them); it is the second-best performer in each of the 3 remaining benchmarks, with a minimal difference with respect to the best performer in 2 of the 3 benchmarks.

We also investigated several mapping properties, both static (invalidity, degeneracy, locality) and dynamic (evolvability, neutrality). Results showed that WHGE exhibits much better properties than GE and πGE; WHGE tends to exhibit better degeneracy, evolvability, and neutrality than SGE while SGE exhibits better locality. Although this analysis does not provide any ultimate answer to the research question of relating the properties of a mapping to the quality of solutions, it does provide useful insights in this respect.

Overall, we believe that the experimental results provide strong indications of the potential of the proposed WHGE mapping.

Acknowledgements

The authors are grateful to the anonymous reviewers for their numerous and insightful comments.

References

[1] C. Ryan, J. Collins, and M. O. Neill, *Grammatical evolution: Evolving programs for an arbitrary language*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 83–96.

[2] M. O'Neill and C. Ryan, "Grammatical evolution," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 349–358, 2001.

[3] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. MIT press, 1992, vol. 1.

[4] I. Dempsey, M. O'Neill, and A. Brabazon, *Foundations in grammatical evolution for dynamic environments*. Springer, 2009, vol. 194.

[5] J. Hugosson, E. Hemberg, A. Brabazon, and M. O'Neill, "An investigation of the mutation operator using different representations in grammatical evolution," in *Proc. 2nd International Symposium Advances in Artificial Intelligence and Applications*, vol. 2, 2007, pp. 409–419.

[6] R. I. Mckay, N. X. Hoai, P. A. Whigham, Y. Shan, and M. O'Neill, "Grammar-based genetic programming: a survey," *Genetic Programming and Evolvable Machines*, vol. 11, no. 3-4, pp. 365–396, 2010.

[7] A. Brabazon, M. O'Neill, and S. McGarraghy, *Natural computing algorithms*. Springer, 2015.

[8] A. O. de la Puente, R. S. Alfonso, and M. A. Moreno, "Automatic composition of music by means of grammatical evolution," *SIGAPL APL Quote Quad*, vol. 32, no. 4, pp. 148–155, Jun. 2002.

[9] E. Medvet, A. Bartoli, and J. Talamini, "Road traffic rules synthesis using grammatical evolution." in *EvoApplications (2)*, 2017, pp. 173–188.

[10] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao, "Syntactical similarity learning by means of grammatical evolution," in *Parallel Problem Solving from Nature – PPSN XIV: 14th International Conference, Edinburgh, UK, September 17-21, 2016, Proceedings*. Cham: Springer International Publishing, 2016, pp. 260–269.

[11] M. Fenton, C. McNally, J. Byrne, E. Hemberg, J. McDermott, and M. O'Neill, "Discrete planar truss optimization by node position variation using grammatical evolution," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 4, pp. 577–589, 2016.

[12] P. B. Miranda and R. B. Prudêncio, "Generation of particle swarm optimization algorithms: An experimental study using grammar-guided genetic programming," *Applied Soft Computing*, 2017.

[13] C. Ryan, A. Azad, A. Sheahan, and M. O'Neill, "No coercion and no prohibition, a position independent encoding scheme for evolutionary algorithms–the chorus system," in *European Conference on Genetic Programming*. Springer, 2002, pp. 131–141.

[14] L. Georgiou and W. J. Teahan, "Constituent grammatical evolution," in *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, vol. 22, no. 1. Citeseer, 2011, p. 1261.

[15] H.-T. Kim and C. W. Ahn, "Umbge: Univariate model based grammatical evolution," *Journal of Computational and Theoretical Nanoscience*, vol. 13, no. 7, pp. 4104–4110, 2016.

[16] M. O'Neill, A. Brabazon, M. Nicolau, S. M. Garraghy, and P. Keenan, *πGrammatical Evolution*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 617–629.

[17] N. Lourenço, F. B. Pereira, and E. Costa, "Sge: a structured representation for grammatical evolution," in *International Conference on Artificial Evolution (Evolution Artificielle)*. Springer, 2015, pp. 136–148.

[18] J. McDermott, D. R. White, S. Luke, L. Manzoni, M. Castelli, L. Vanneschi, W. Jaskowski, K. Krawiec, R. Harper, K. De Jong *et al.*, "Genetic programming needs better benchmarks," in *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. ACM, 2012, pp. 791–798.

[19] 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, pp. 3–29, 2013.

[20] F. Rothlauf and D. E. Goldberg, "Redundant representations in evolutionary computation," *Evolutionary Computation*, vol. 11, no. 4, pp. 381–415, 2003.

[21] F. Rothlauf, "Representations for genetic and evolutionary algorithms," in *Representations for Genetic and Evolutionary Algorithms*. Springer Berlin Heidelberg, 2006, pp. 9–32.

[22] F. Rothlauf and M. Oetzel, "On the locality of grammatical evolution," in *European Conference on Genetic Programming*. Springer, 2006, pp. 320–330.

[23] D. Wilson and D. Kaur, "Search, neutral evolution, and mapping in evolutionary computing: A case study of grammatical evolution," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 3, pp. 566–590, 2009.

[24] T. Castle and C. G. Johnson, "Positional effect of crossover and mutation in grammatical evolution," in *European Conference on Genetic Programming*. Springer, 2010, pp. 26–37.

[25] M. B. Correia, "A study of redundancy and neutrality in evolutionary optimization," *Evolutionary computation*, vol. 21, no. 3, pp. 413–443, 2013.

[26] A. Thorhauer and F. Rothlauf, "On the locality of standard search operators in grammatical evolution," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2014, pp. 465–475.

[27] A. Thorhauer, "On the non-uniform redundancy in grammatical evolution," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2016, pp. 292–302.

[28] E. Medvet, "A comparative analysis of dynamic locality and redundancy in grammatical evolution," in *Genetic Programming: 20th European Conference, EuroGP 2017, Amsterdam, Netherlands, April 19-21, 2017,*

Proceedings. Cham: Springer International Publishing, 2017, p. to appear.

[29] ——, "Hierarchical grammatical evolution," in *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO*, 2017.

[30] D. Fagan, M. O'Neill, E. Galván-López, A. Brabazon, and S. McGarraghy, "An analysis of genotype-phenotype maps in grammatical evolution," in *European Conference on Genetic Programming*. Springer, 2010, pp. 62–73.

[31] N. Lourenço, F. B. Pereira, and E. Costa, "Unveiling the properties of structured grammatical evolution," *Genetic Programming and Evolvable Machines*, pp. 251–289, 2016.

[32] N. Lourenço, J. Ferrer, F. B. Pereira, and E. Costa, "A comparative study of different grammar-based genetic programming approaches," in *European Conference on Genetic Programming*. Springer, Cham, 2017, pp. 311–325.

[33] L. Altenberg, "Probing the axioms of evolutionary algorithm design: Commentary on "on the mapping of genotype to phenotype in evolutionary algorithms" by peter a. whigham, grant dick, and james maclaurin," *Genetic Programming and Evolvable Machines*, pp. 1–5, 2017.

[34] E. Medvet and A. Bartoli, "On the automatic design of a representation for grammar-based genetic programming," in *Genetic Programming*, M. Castelli, L. Sekanina, M. Zhang, S. Cagnoni, and P. García-Sánchez, Eds. Cham: Springer International Publishing, 2018, pp. 101–117.

[35] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao, "Active learning of regular expressions for entity extraction," *IEEE Transactions on Cybernetics*, 2017.

[36] ——, "Inference of regular expressions for text extraction from examples," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 5, pp. 1217–1230, 2016.

[37] N. J. Radcliffe and P. D. Surry, "Fitness variance of formae and performance prediction," in *FOGA*, vol. 3, 1994, pp. 51–72.

[38] M. O'Neill and C. Ryan, "Genetic code degeneracy: Implications for grammatical evolution and beyond," in *European Conference on Artificial Life*. Springer, 1999, pp. 149–153.

[39] E. Medvet, F. Daolio, and D. Tagliapietra, "Evolvability in grammatical evolution," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '17. New York, NY, USA: ACM, 2017, pp. 977–984.

[40] E. Medvet and T. Tušar, "The du map: A visualization to gain insights into genotype-phenotype mapping and diversity," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '17. New York, NY, USA: ACM, 2017, pp. 1705–1712.

[41] J. A. Walker and J. F. Miller, "The automatic acquisition, evolution and reuse of modules in cartesian genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 4, pp. 397–417, 2008.

[42] L. Vanneschi, M. Castelli, and L. Manzoni, "The k landscapes: a tunably difficult benchmark for genetic programming," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. ACM, 2011, pp. 1467–1474.

[43] M. Keijzer, "Improving symbolic regression with interval arithmetic and linear scaling," *Genetic programming*, pp. 275–299, 2003.

[44] N. Q. Uy, N. X. Hoai, M. O'Neill, R. I. McKay, and E. Galván-López, "Semantically-based crossover in genetic programming: application to real-valued symbolic regression," *Genetic Programming and Evolvable Machines*, vol. 12, no. 2, pp. 91–119, 2011.

[45] L. Pagie and P. Hogeweg, "Evolutionary consequences of coevolving targets," *Evolutionary computation*, vol. 5, no. 4, pp. 401–418, 1997.

[46] E. J. Vladislavleva, G. F. Smits, and D. Den Hertog, "Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 333–349, 2009.

[47] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao, "Gomge: Genepool optimal mixing on grammatical evolution," in *International Conference on Parallel Problem Solving from Nature*, 2018, to appear.

[48] E. Medvet, A. Bartoli, A. De Lorenzo, and F. Tarlao, "Designing automatically a representation for grammatical evolution," *Genetic Programming and Evolvable Machines*, pp. 1–29, 2018.

[49] M. Nicolau, "Understanding grammatical evolution: initialisation," *Genetic Programming and Evolvable Machines*, Jul 2017.

[50] M. Pawlik and N. Augsten, "Efficient computation of the tree edit distance," *ACM Transactions on Database Systems (TODS)*, vol. 40, no. 1, p. 3, 2015.

[51] M. Tomassini, L. Vanneschi, P. Collard, and M. Clergue, "A study of fitness distance correlation as a difficulty measure in genetic programming," *Evolutionary Computation*, vol. 13, no. 2, pp. 213–239, 2005.

[52] H. Mengistu, J. Lehman, and J. Clune, "Evolvability search: Directly selecting for evolvability in order to study and produce it," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, ser. GECCO '16. New York, NY, USA: ACM, 2016, pp. 141–148.

[53] E. Galván-López and R. Poli, "An empirical investigation of how and why neutrality affects evolutionary search," in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. ACM, 2006, pp. 1149–1156.

[54] W. Banzhaf and A. Leier, "Evolution on neutral networks in genetic programming," *Genetic programming theory and practice III*, pp. 207–221, 2006.

[55] E. Galván-López, R. Poli, A. Kattan, M. O'Neill, and A. Brabazon, "Neutrality in evolutionary algorithms… what do we know?" *Evolving Systems*, vol. 2, no. 3, pp. 145–163, Sep 2011.

[56] G. Lu, J. Li, and X. Yao, "Fitness-probability cloud and a measure of problem hardness for evolutionary algorithms," in *European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer, 2011, pp. 108–117.

Alberto Bartoli received the degree in Electrical Engineering in 1989 cum laude and the PhD degree in Computer Engineering in 1993, both from the University of Pisa, Italy. Since 1998 he is an Associate Professor at the Department of Engineering and Architecture of University of Trieste, Italy, where he is the Director of the Machine Learning Lab. His research interests include machine learning applications, evolutionary computing, and security.

Mauro Castelli obtained his Master degree in Computer Science at the University of Milano Bicocca (Italy) in 2008 ("summa cum Laude"), and his PhD at the University of Milano Bicocca in 2012. He is assistant professor at NOVA IMS, Universidade Nova de Lisboa (Portugal). His main research interests are in the field of Artificial Intelligence (in particular, Evolutionary Computation and Genetic Programming) and in the application of Machine Learning techniques to solve complex real-life problems, especially in the field of biology and medicine.

Eric Medvet received the degree in Electronic Engineering cum laude in 2004 and the PhD degree in Computer Engineering in 2008, both from the University of Trieste, Italy. He is currently an Assistant Professor in Computer Engineering at the Department of Engineering and Architecture of University of Trieste, Italy. His research interests include Genetic Programming and Machine Learning applications, in particular concerning Android malware detection and information retrieval.