On the Automatic Design of a Representation for Grammar-based Genetic Programming

Eric Medvet and Alberto Bartoli

Department of Engineering and Architecture, University of Trieste, Trieste, Italy
emedvet@units.it, bartoli.alberto@units.it

Abstract. A long-standing problem in Evolutionary Computation consists in how to choose an appropriate representation for the solutions. In this work we investigate the feasibility of synthesizing a representation *automatically*, for the large class of problems whose solution spaces can be defined by a context-free grammar. We propose a framework based on a form of meta-evolution in which individuals are candidate representations expressed with an ad hoc language that we have developed to this purpose. Individuals compete and evolve according to an evolutionary search aimed at optimizing such representation properties as redundancy, locality, uniformity of redundancy.
We assessed experimentally three variants of our framework on established benchmark problems and compared the resulting representations to human-designed representations commonly used (e.g., classical Grammatical Evolution). The results are promising in the sense that the evolved representations indeed exhibit better properties than the human-designed ones. Furthermore, while those improved properties do not result in a systematic improvement of search effectiveness, some of the evolved representations do improve search effectiveness over the human-designed baseline.

Keywords: Genotype-phenotype mapping, Grammatical Evolution, Meta-evolution

1 Introduction

The choice of the representation of individuals in an Evolutionary Algorithm (EA) has been a central point in the field of Evolutionary Computation since its inception [29, 22]. In many cases, that choice has been guided *a priori* by analogies with the biology, in which researchers looked for inspiration while designing their artificial evolutionary systems, on the assumption that Nature eventually succeeded as an effective search method [34]. On the other hand, the impact of the representation on the EA search effectiveness has also been widely studied *a posteriori*. In this respect, a common and well established practice consists in investigating any possible relationship between properties of the representation such as, e.g., redundancy and locality [23, 15, 31], and higher level properties of the EA, e.g., neutrality [3] and evolvability [14].

Despite these efforts, it is fair to claim that both approaches (a priori and a posteriori) failed in clearly determining if and when a representation can guarantee the search effectiveness of an EA: copying from the Nature does not necessarily lead to a good design [30, 7] and there is not a clear view of which properties actually explain a good or a poor search effectiveness [1]. Indeed, the debate is still lively, with arguments ranging from (deemed) misuse of Nature analogies [35] to experimental-based (counter-)evidences [24] and outcomes including guidelines for the design of a representation [34] or directions for future research [29].

A case of particular interest is the one of indirect representations, i.e., those in which each individual is represented by means of a *genotype* and a *phenotype* and a mapping function exists for mapping the former to the latter. Practical motivations for choosing an indirect representation include the possibility of using standard genetic operators—whose behavior is well known—and, at the same time, tackling problems for which specific constraints act on the solutions (i.e., phenotypes). Moreover, indirect representations do have a counterpart in biology, where the form of living organisms depends on the result of a transcription process operating on encoded genetic material. Finally, indirect representation properties can be easily defined and studied both analytically and experimentally basing on the mapping function.

One of the most used EAs based on an indirect representation is Grammatical Evolution (GE) [25], a form of grammar-based Genetic Programming (GP) [10], which captures all the three aspects of indirect representations described above. First, GE allows tackling the large class of problems in which constraints on the solutions may be expressed by means of a context-free grammar (CFG). Second, according to its inventors, the overall GE framework was directly inspired by Nature [17]. Third, the properties of the GE genotype-phenotype mapping function have been widely studied [32, 31, 11]: indeed, those properties eventually served as main goals while designing new GE variants, essentially consisting in new mapping functions which were shown to be more effective than the original approach [9, 12].

In this work, we attempt to provide new insights on the long-standing, undercurrent topic of the choice of the representation. To this end, we consider the broad class of EAs corresponding to grammar-based GP and propose a novel approach for the automatic design of a representation driven by an evolutionary search aimed at optimizing the representation properties. Our proposal thus tries, in a sense, to merge the a priori and a posteriori approaches.

Our contribution consists of the following: (a) we define a class of representations in which the genotype is a variable-length bit string and the phenotype is a valid string w.r.t. a user-provided grammar; (b) we propose an evolutionary framework for searching the aforementioned space of representations; (c) we experimentally investigate the ability of the proposed framework to generate representations whose properties and search effectiveness are better than existing, established representations.

In detail, the class of representations is defined by a genotype-phenotype mapping function template whose variable parts are described with a language which we defined by means of a CFG. The mapping function template and the language are such that: (i) any representation in the resulting class is a valid genotype-phenotype mapping function—i.e., any input bit string is mapped to a valid phenotype in a finite number of steps; (ii) it is possible to express such existing and established representations as the original GE mapping [25] and the recently proposed HGE and WHGE [12]. Having defined the search space in terms of a CFG, we use a grammar-based evolutionary search method (CFG-GP [36]) which we augmented using a diversity promotion strategy in order to improve search effectiveness [13]. For driving the search, we use a fitness function measuring to which degree an individual (i.e., a genotype-phenotype mapping function) exhibits such mapping properties as redundancy, locality, uniformity of redundancy. We compute those measures on a large amount of mappings obtained from a sample grammar and a set of randomly generated genotypes.

We investigated 3 search variants differing in the fitness definition and optimization strategy (i.e., single-objective vs. multi-objective). We assessed each obtained representation experimentally not only in terms of the mapping properties, but also in terms of higher level EA properties (diversity) and of the search effectiveness achieved on a small set of benchmark problems previously used in the literature for assessing GE and its variants. The results are promising as some of the automatically generated representations are better than the existing ones. Although our findings do not imply that automatically-designed representations may fully surrogate carefully human-designed representations, they further corroborate the importance of representation properties and might ignite new research in the novel field of "self-evolving" evolutionary algorithms.

The remainder of the paper is organized as follows. In Section 2, we briefly survey the state-of-the-art. In Section 3, we introduce our genotype-phenotype mapping function template and the related CFG for describing its variable parts. In Section 4, we describe which are the properties we use to drive the evolution of the mapping function and how we compute them. In Section 5, we present and discuss the results of our experimental evaluation. Finally, in Section 6, we draw the conclusions.

2 Related work

Broadly speaking, our proposal is a form of meta-evolution [6] (also known as hyper-heuristic [20] or self-adaptation [26]), where parts of an EA are chosen or tuned according to a second-level evolutionary search. In most cases, the literature focuses on specific EA parameters which can be optimized, rather than designed from scratch—e.g., mutation and crossover rate in Genetic Semantic Programming [2] or trial vector and control parameters in Differential Evolution [21]. The application of evolutionary computation to evolve (online or offline) components, rather than parameter values, of an EA is instead still believed to be in its infancy [29], in particular for representation and variation

operators. For the former, the scarcity of research results may be explained by its hardness, as observed by De Jong [5]: "perhaps the most difficult and least understood area of EA design is that of adapting its internal representation."

Concerning the evolution of operators, the authors of [8] show how they evolved a general purpose mutation operator for Evolutionary Programming which outperforms existing operators on classes of functions (i.e., problems); they also experimentally show that a mutation operator evolved for a specific problem is better than a general purpose evolved operator. A similar goal is aimed at in [4], where a framework for the online evolution of the operators, together with the solutions, is proposed: as in the previously cited work, operators are represented as trees and evolved using GP. Similarly to the present work, [4] considers also other EA properties (diversity) other than search effectiveness as a criterion of analysis.

Concerning the automatic design or adaptation of representations, a proposal is presented in [28], where genotype-phenotype mapping for continuous optimization problems is considered. The authors show, using a proof-of-concept self-adaptation mechanism, that feed-forward neural networks can be used to represent and improve a genotype-phenotype mapping, also for problems of realistic complexity. Similarly to our work, the authors carefully consider redundancy and locality in their analysis.

Another view on automatic design of representation is given by [27], which again addresses the class of real-valued optimization problems: here, the representation is the way in which the real values are encoded using a bit string. With the premise that they focused only on (few) synthetic problems, due to the high computational costs implied by meta-evolution, the authors find that an evolved representation may improve the classical Gray encoding.

Also relevant w.r.t. our work are some proposals concerning grammar-based GP in which the grammar itself is evolved (or improved) online, during the evolution [38, 18]. Despite the evolution of a new, general purpose representation was not among the goals of the cited papers (they rather attempt to discover more knowledge about the problem defined by the user-provided grammar by improving the grammar itself), they somehow demonstrate how a representation can change while still enforcing the problem-specific constraints on the solutions. In conclusion, to the best of our knowledge, our work is the first attempt of evolving a general purpose representation for a large class of problems, as the one addressable with grammar-based GP.

3 Representation template

We consider a family of EAs with an indirect representation where the *genotype* $\hat{g}$ is a variable-length bit string and the *phenotype* $\hat{p}$ is a string of a language $\mathcal{L}(\mathcal{G})$ defined by a CFG $\mathcal{G} = (N, T, s_0, R)$, 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. We do not pose any constraint on components of the EA other than the representation (e.g., selection criteria for

reproduction of removal of individuals, initialization). It is worth to note that many significant and widely used variants of GE (beyond its original version) belong to this family of EAs (e.g., πGE [16], HGE and WHGE [12]).

We define a *representation template*, i.e., a template of a mapping between a variable-length bit string (genotype) and a string in $\mathcal{L}(\mathcal{G})$ (phenotype), as follows. The mapping is based on the notion of *derivation tree* of a symbol s in $N \cup T$. Such a tree is rooted at s and the children of each non-terminal node $s' \in N$ are symbols (in the proper order) of one of the derivation options for s' in $\mathcal{G}$. The derivation tree is constructed with the algorithm specified below. The mapping occurs in two steps: the input genotype $\hat{g}$ is mapped to a derivation tree of the initial symbol s_0 of $\mathcal{G}$; the corresponding phenotype $\hat{p}$ is then obtained by concatenating, from the left to the right, the leaf nodes of the derivation tree.

Construction of a derivation tree is performed by a function $\textsc{Map}(s, g, d)$, where s is a symbol of $T \cup N$, g is a bit string, and $d \in \mathbb{N}^+ \cup \{0\}$ is a positive number. This function essentially consists in three key steps: (i) choose one derivation option among the ones available for s, by invoking function Choose(); (ii) obtain from g several bit strings, by invoking function Divide(); (iii) recursively call itself for each symbol in the chosen derivation option, with the symbol, one of the bit strings previously obtained, a counter $d + 1$ of recursion depth as input parameters.

Functions Choose() and Divide() are *parameters* of Map() and their signature includes a bit string as input argument. Their domain consists of all the functions that can be defined by a language described in Section 3.1 that we developed. The search space for representations, thus, essentially consists in all the possible implementations for Choose() and Divide().

The mapping of $\hat{g}$ to a derivation tree of s_0 is done by invoking $\textsc{Map}(s_0, \hat{g}, 0)$. The corresponding phenotype $\hat{p}$ is then obtained by concatenating the leaf nodes of the derivation tree.

In details, Map() is shown in Algorithm 1 and works as follows. If s is a terminal node, the tree composed by a single node s is returned by $\textsc{Map}(s, g, d)$, regardless of the values of g and d. Otherwise, the following steps are performed.

1. The derivation rule r_s for the input argument s is obtained.
2. A vector $\boldsymbol{e} \in \mathbb{R}^{|r_s|}$ is built, where each element e_j is the product of the *expressiveness* of all the symbols in the jth option of r_s. The expressiveness of a symbol s' (denoted by $\textsc{Expressiveness}(s')$ in Algorithm 1) is a measure of the expressive power of s': we quantify expressiveness with the number of different derivation trees which can be obtained from s'. We limit the counting to derivation trees with a maximum d_{expr} depth (an implicit parameter of Expressiveness() and hence of the representation itself) in order to cope with non-finite languages, for which $\textsc{Expressiveness}(s')$ may be infinite.
3. If the input argument d is greater than or equal to a predefined value $d_{\max}$ (a parameter of the representation), the index i of the chosen rule option is set to the value for which e_i is the lowest in $\boldsymbol{e}$. Otherwise, i is set to the return value of a function Choose() which takes as input $g, \boldsymbol{e}, d$ and returns

Algorithm 1 The genotype-phenotype recursive mapping function, which is first invoked as $\textsc{Map}(s_0, \hat{g}, 0)$.

function $\textsc{Map}(s, g, d)$
  $t \leftarrow \textsc{TreeNode}(s)$
  if $s \in N$ **then** ▷ s is a non-terminal
    $r_s \leftarrow \textsc{RuleFor}(s)$
    for $j \in \{1, \dots, |r_s|\}$ **do**
      $e_j \leftarrow \prod_{s' \in \textsc{Symbols}(r_s, j)} \textsc{Expressiveness}(s')$
    end for
    $\boldsymbol{e} \leftarrow (e_1, \dots, e_{|r_s|})$
    if $d \geq d_{\max}$ **then** ▷ maximum depth reached
      $i \leftarrow \arg\min_{j \in \{1, \dots, |r_s|\}} e_j$
    else
      $i \leftarrow \textsc{Choose}(g, \boldsymbol{e}, d)$
    end if
    $(s_1, \dots, s_n) \leftarrow \textsc{Symbols}(r_s, i)$
    for $j \in \{1, \dots, n\}$ **do**
      $e_j \leftarrow \textsc{Expressiveness}(s_j)$
    end for
    $\boldsymbol{e} \leftarrow (e_1, \dots, e_n)$
    $(g_1, \dots, g_m) \leftarrow \textsc{Divide}(g, \boldsymbol{e}, d)$
    for $j \in \{1, \dots, n\}$ **do** ▷ Append children
      $\textsc{AppendChild}(t, \textsc{Map}(s_j, g_j, d + 1))$
    end for
  end if
  return t
end function

a number that will be used at the next step for choosing one of the options of the derivation rule r_s.

4. The sequence of symbols $s_1, \dots, s_n$ corresponding to the ith option of the r_s rule is obtained. We denote by $\textsc{Symbols}()$ the corresponding grammar look-up function in Algorithm 1; $\textsc{Symbols}()$ is protected, i.e., it works for any i by using $\min(|r_s| - 1, \max(0, \lfloor i \rfloor))$ instead of the original argument i.
5. The vector $\boldsymbol{e}$ is reset to $(e_1, \dots, e_n)$, where e_j is the expressiveness of s_j obtained at the previous step.
6. A sequence $(g_1, \dots, g_m)$ of bit strings is set to the return value of a function $\textsc{Divide}()$ which takes as input $g, \boldsymbol{e}, d$ and returns a sequence of bit strings. Each of these bit strings will be used at the next step for constructing subtrees to be appended to the derivation tree being constructed.
7. For each symbol s_j in $s_1, \dots, s_n$, the tree obtained by recursively invoking the $\textsc{Map}(s_j, g_j, d + 1)$ is appended to the tree (initially) composed of the only node s, which is eventually returned. While performing this step, in case $j > m$ (i.e., if there are fewer bit strings than symbols to built the children of s), an empty bit string is passed to $\textsc{Map}()$ as g_j.

```
⟨mapper⟩ ::= ⟨n⟩ ⟨lg⟩
    ⟨n⟩ ::= ⟨const.n⟩ | ⟨var.n⟩ | ⟨fun.n.g⟩ ( ⟨g⟩ ) | ⟨fun.n.n,n⟩ ( ⟨n⟩ , ⟨n⟩ ) | ⟨fun.n.ln⟩ ( ⟨ln⟩ ) |
            ⟨fun.n.ln,n⟩ ( ⟨ln⟩ , ⟨n⟩ ) | ⟨fun.n.lg⟩ ( ⟨lg⟩ )
   ⟨ln⟩ ::= ⟨var.ln⟩ | ⟨fun.ln.n⟩ ( ⟨n⟩ ) | ⟨fun.ln.n,n⟩ ( ⟨n⟩ , ⟨n⟩ ) | apply ( ⟨fun.n.g⟩ , ⟨lg⟩ )
    ⟨g⟩ ::= ⟨var.g⟩ | ⟨fun.g.g,n⟩ ( ⟨g⟩ , ⟨n⟩ ) | ⟨fun.g.lg,n⟩ ( ⟨lg⟩ , ⟨n⟩ )
   ⟨lg⟩ ::= ⟨fun.lg.g,n⟩ ( ⟨g⟩ , ⟨n⟩ ) | ⟨fun.lg.g,ln⟩ ( ⟨g⟩ , ⟨ln⟩ ) | apply ( ⟨fun.g.g,n⟩ , ⟨ln⟩ , ⟨g⟩ )
⟨const.n⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
  ⟨var.n⟩ ::= depth | g.count.r | g.count.rw
  ⟨var.g⟩ ::= g
 ⟨var.ln⟩ ::= ln
⟨fun.n.g⟩ ::= size | weight | weight.r | int

⟨fun.n.n,n⟩ ::= + | - | * | / | %
 ⟨fun.n.ln⟩ ::= length | max.index | min.index
⟨fun.n.ln,n⟩ ::= get
 ⟨fun.n.lg⟩ ::= length
 ⟨fun.ln.n⟩ ::= seq
⟨fun.ln.n,n⟩ ::= repeat
⟨fun.g.g,n⟩ ::= rotate.left | rotate.right | substring
⟨fun.g.lg,n⟩ ::= get
⟨fun.lg.g,n⟩ ::= split | repeat
⟨fun.lg.g,ln⟩ ::= split.w
```

Fig. 1. The CFG $\mathcal{G}_{\text{MAP}}$ defining the language for the CHOOSE() and DIVIDE() functions and hence for an instance of the genotype-phenotype mapping function template defined by MAP().

Regardless of the actual behavior of CHOOSE() and DIVIDE(), it can be easily seen that MAP() always returns a derivation tree (from which a valid phenotype is then obtained) in a finite number of steps. First, whenever the value of d (which is increased at each recursive invocation) reaches a threshold, the derivation option is chosen as the one with the lowest expressiveness, instead of by using the CHOOSE() function: since in any valid CFG, for any non-terminal symbol, there is at least one derivation option with a finite expressiveness, this guarantees that in a finite number of steps MAP() will be invoked with a terminal symbol $s \in T$. Second, regardless of the return value of CHOOSE(), a valid derivation is always chosen for s, since only options of r_s are considered.

3.1 Language for the mapping function

Functions CHOOSE() and DIVIDE() are parameters of the mapping function. The space of possible values for these parameters consists of all the functions that may be described by the CFG $\mathcal{G}_{\text{MAP}}$ specified in Figure 1 and discussed below.

$\mathcal{G}_{\text{MAP}}$ includes terminal symbols representing numerical constants ($0, \ldots, 9$), input arguments (g for g, ln for $\boldsymbol{e}$, and depth for d), and functions (e.g., size returns the length of a bit string, weight returns the number of bits set to 1 in a bit string).

Names for the non-terminal symbols representing functions begin with fun and encode the signature of the function with a simple conventional rule. For example, ⟨fun.lg.g,n⟩ represents functions whose return value is of type ⟨lg⟩ and whose list of input arguments is of types ⟨g⟩ and ⟨n⟩. Non-terminal symbols other than functions represent data types: ⟨n⟩ represents numbers, ⟨ln⟩ represents sequences of numbers, ⟨g⟩ represents bit strings, ⟨lg⟩ represents sequences of bit strings. Thus, the above example represents functions whose return value is a number and that take a bit string followed by a number as input arguments.

Concerning terminal symbols that represent functions, as the size and weight described above, we omit a detailed description of the semantics, leaving it implicit in the name of the corresponding symbols. All the functions are type-protected, i.e., they guarantee that a correctly typed value is always returned—

e.g., the number n in which a bit string g is split by the split function is internally adjusted as $\min(\ell(g), \max(1, \lfloor n \rfloor))$, where $\ell(g)$ is the length of the bit string g.

Symbols g.count.r and g.count.rw corresponds to accessing a global counter, the former reads the value of the counter while the latter reads and then increments its value. By "global" we mean that a single counter is maintained during the execution of both CHOOSE() and DIVIDE(); this counter is set to 0 when the enclosing MAP() is first called with parameters $s_0, \hat{g}, 0$. Including a global counter allows to express also genotype-phenotype mapping functions which are not inherently recursive, but can be expressed as recursive function thanks to the counter: the original GE mapping fits this case (see Figure 2).

Non-terminal symbol ⟨mapper⟩ is the crucial component for expressing an instance of the genotype-phenotype mapping function, i.e., of functions CHOOSE() and DIVIDE(). This symbol can be derived only as a pair ⟨n⟩, ⟨lg⟩: the concatenation of the leaves of the derivation tree rooted at the left child of ⟨mapper⟩ is the function CHOOSE(); similarly, the right child represents the function DIVIDE().

As stated in the introduction, a key feature of our proposal is that it allows expressing such existing and established genotype-phenotype mapping functions as those used in GE, HGE, and WHGE. Indeed, Figure 2 shows the CHOOSE() and DIVIDE() functions corresponding to (a slightly improved version of) GE and WHGE. Differently than the original GE mapping, this version does not require a mechanism for aborting the mapping when it looks endless (in [25] there is a maximum number of genotype reuses, i.e., wrappings), since that case is addressed by comparing d against $d_{\max}$ in MAP().

GE	CHOOSE()	int(substring(rotate.left(g, *(gl.count.rw, 8)), 8))
	DIVIDE()	repeat(g, length(ln))
WHGE	CHOOSE()	max.index(apply(weight.r, split(g, length(ln))))
	DIVIDE()	split.w(g, ln)

Fig. 2. CHOOSE() and DIVIDE() for the original GE mapping and for WHGE.

4 Properties-driven evolution

Since we defined the search space of the problem of the automatic design of a representation by means of the CFG $\mathcal{G}_{\text{MAP}}$, we can tackle that problem using any grammar-based GP approach (e.g., GE, πGE, SGE, HGE, WHGE, CFG-GP), provided that we define a fitness function suitable for driving the search. In this work, we want a fitness function able to capture the degree to which a candidate representation $m \in \mathcal{L}(\mathcal{G}_{\text{MAP}})$ exhibits the desired mapping properties.

Among the several properties of indirect representations which have been studied in the literature (see [22] for a comprehensive analysis), we considered redundancy, locality, and uniformity of redundancy—we actually considered non-locality and non-uniformity in order to conform to the semantics of "the lower, the better".

We measure the properties of a representation m basing on how m maps a predefined set G of genotypes to a corresponding set P of phenotypes using a predefined CFG $\mathcal{G}_{\text{learn}}$. That is, for each $\hat{g} \in G$ we construct $\hat{p} = m(\hat{g})$ by

concatenating, from the left to the right, the leaf nodes of the derivation tree returned by $\textsc{Map}(\hat{g}, s_0, 0)$, where $\textsc{Map}()$ is the instance of the map function template corresponding to m and s_0 is the starting symbol of $\mathcal{G}_{\text{learn}}$. Having constructed P from G according to m, we quantify the properties of interest as follows.

The *redundancy* of m is measured as $1 - \frac{|P|}{|G|}$, i.e., one minus the ratio between the number $|P|$ of unique phenotypes and the number $|G|$ of unique genotypes.

The *locality* of m is measured as the Pearson correlation between the distances among genotypes and distances among phenotypes. More formally, let D^G be the sequence of $\frac{|G|(|G|-1)}{2}$ genotype distances (i.e., $d^G_{i,j} = d^G(\hat{g}_i, \hat{g}_j)$ is the distance between the ith and the jth elements of G, with $j < i$) and let D^P be the corresponding sequence of phenotype distances (i.e., $d^P_{i,j} = d^P(m(\hat{g}_i), m(\hat{g}_j))$). The locality is the Pearson correlation $\text{cor}(D^G, D^P)$ between D^G and D^P. As distances, we used the edit distance for both bit strings and strings of $\mathcal{L}(\mathcal{G}_{\text{learn}})$. The non-locality is measured as $1 - \frac{1+\text{cor}(D^G, D^P)}{2}$, such that it is 0 when genotype and phenotype distances are perfectly correlated ($\text{cor}(D^G, D^P) = 1$), and 1 when they are inversely correlated ($\text{cor}(D^G, D^P) = -1$).

Finally, the *uniformity* of m is measured by means of the coefficient of variation of the size of the partitions of G for which every genotype in the partition corresponds to the same phenotype. More formally, let $G_1, \ldots, G_{|P|}$ be the partitions of G such that, for each k, $\forall \hat{g}_i, \hat{g}_j \in G_k : m(\hat{g}_i) = m(\hat{g}_j)$, and let $S = |G_1|, \ldots, |G_{|P|}|$ contains the sizes of the partitions. The non-uniformity is the coefficient of variation $\frac{\sigma_S}{\mu_S}$ of S.

In order to define a criterion for driving the evolutionary search in the space of representations, we considered that, according to many studies, redundancy, locality, and uniformity appears to affect the effectiveness of the search in the respective order [17, 23, 14]. We hence explored three variants for driving the search for a representation: by minimizing redundancy only (single-objective), by minimizing redundancy and non-locality (multi-objective), and by minimizing redundancy, non-locality, and non-uniformity (multi-objective). We denote the respective *search variants* by R, R/NL, and R/NL/NU.

In all of our experiments, we used CFG-GP [36] as the evolutionary search algorithm, in a version augmented with the diversity promotion mechanism presented in [13] (with $n_{\text{partition}} = 10$ as partition size, phenotype equivalence as partitioning criterion, and youngest individual as parent representative selection criterion) and with the selection criteria for reproduction and removal of individuals based on the comparison between individuals according to the Pareto dominance.

5 Experiments and discussion

We performed an experimental evaluation aimed at answering the two following research questions: RQ1: Can we evolve a representation which is better than the existing ones in terms of redundancy, locality, and uniformity? RQ2: Are the evolved representations also effective when used inside an actual EA?

In order to answer RQ1, we proceeded as follows. First, we executed a number $n_{\text{run}}^{\text{learning}} = 10$ of *learning runs* for each of our proposed variants R, R/NL, R/NL/NU. From each learning run we obtained a set of non-dominated representations (R/NL and R/NL/NU variants, multi-objective) and a set of representations with the same, minimal redundancy value (R variant, single-objective).

Second, for each learning run, we selected a subset of $n_{\text{repr}}^{\text{validation}} = 5$ representations for further analysis, as follows. We selected one representation randomly and then we selected iteratively, one at once, the $n_{\text{repr}}^{\text{validation}} - 1$ representations which are farthest from those already selected in terms of Euclidean distance on the fitness space (in case of ties we chose one representation at random).

Third, for each selected representation m, we performed a number $n_{\text{run}}^{\text{validation}} = 5$ of *validation runs* on each of the three validation problems specified below. That is, we solved each of those problems with representation m and the evolutionary search algorithm resulting from Table 1 (right).

In summary, we performed $3 \times 10 = 30$ learning runs and $3 \times 10 \times 5 \times 3 \times 5 = 2250$ validation runs. The software we developed for this experimentation is publicly available[1].

Table 1. Parameters for the evolutionary runs.

	Learning	Validation
Population size	500	500
Pop. initialization	Ramped half-and-half	Random
Generations	50	30
Max depth d_{max}	14	9
Expressiveness depth d_{expr}	N. A.	2
Genotype size	N. A.	1024
Crossover rate	0.8	0.8
Crossover operator	CFG-GP crossover	two-points same length
Mutation rate	0.2	0.2
Mutation operator	CFG-GP mutation	bit flip w. $p_{\text{mut}} = 0.01$
Selection for reproduction	tournament with size 3	tournament with size 3
Selection for removal	worst individual	worst individual
Replacement	$m + m$ w. overlapping	$m + m$ w. overlapping

We structured learning runs as follows. We composed the set of genotypes G with the following steps: (i) we randomly generated a seed set of 10 bit strings, each of length equal to 256 bit; and, (ii) for each genotype in the seed set, we obtained other 9 genotypes by iteratively applying the bit-flip mutation operator (with $p_{\text{mut}} = 0.01$). The rationale was to obtain a uniform distribution of distances among the genotypes, useful in particular for measuring of the locality property. We used the CFG of the Pagie1 problem as grammar $\mathcal{G}_{\text{learn}}$ for mapping G to the corresponding set P of phenotypes. We set the parameters of the evolutionary search with CFG-GP in the space of representations as in Table 1 (left).

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

We used the following three benchmarks as validation problems: the K-Landscape synthetic problem [33] (with $k = 5$), the Pagie1 symbolic regression problem [19], and the Text generation synthetic problem [11]. Two of these benchmarks have been recommended as standard benchmarks for GP performance evaluation [37], whereas the last one (Text) has been designed specifically for assessing GE and presents a grammar of larger complexity.

Table 2 shows the property values for the evolved representations, averaged across the 5 selected representations for each of the 10 learning runs. The first three rows correspond to property values computed in the learning runs only (hence using the Pagie1 grammar only); the second three rows correspond to property values computed using the grammars of the 3 validation problems; the last three rows correspond to property values for the GE, HGE, and WHGE representations computed using the grammars of the 3 validation problems and can be used as *baseline*. We emphasize that all the baseline are *human-designed*, i.e., they are the result of dedicated research efforts.

Table 2. Representation properties.

	Search variant	Redundancy	Non-locality	Non-uniformity
Learn.	R	0		
	R/NL	0.095	0.032	
	R/NL/NU	0.797	0.319	0.077
Val.	R	0.266	0.291	0.284
	R/NL	0.247	0.28	0.292
	R/NL/NU	0.261	0.29	0.288
	GE	0.990	1.000	0.000
	HGE	0.620	0.403	2.211
	WHGE	0.410	0.412	2.689

It can be seen that, in general, property values for the evolved representation are much better than for the baselines. Furthermore, property values on the validation problems appear to be independent from the search variant (R vs. R/NL vs. R/NL/NU). We interpret this result as a combination of: (i) these values are computed on 3 grammars w.r.t. the one used for learning; and, (ii) for multi-objective fitness variants, the shown values tend to "average" different representations, i.e., points which are far away from each other in the fitness space.

In order to answer RQ2 we then examined the search effectiveness of the evolved representations. That is, we examined the fitness values for each of the validation problems when solved with the evolved representations and when solved with the baseline representations. Table 3 shows, for each validation problem, the final best fitness BF and the difference ΔBF between the final and initial best fitness—both BF and ΔBF are averaged, for each evolved representation and baseline, across the $n_{\text{run}}^{\text{validation}} = 5$ validations runs. Index ΔBF is relevant as it should capture the ability of the representation to actually improve the so-

lution during the evolution. For each of the three search variants, Table 3 shows BF and ΔBF obtained with the best, mean, and worst representations among the $n_{\text{run}}^{\text{learning}} = 10$ learning runs with that search variant.

Table 3. Final best fitness BF and difference ΔBF between final and initial best fitness.

		BF			ΔBF		
	Search variant	Best	Mean	Worst	Best	Mean	Worst
KLand.-5	R	0.11	0.6	0.81	0.48	0.11	0
	R/NL	0.58	0.66	1	0.27	0.11	0
	R/NL/NU	0.55	0.7	1	0.33	0.06	0
	GE	1			0		
	HGE	0.58			0.06		
	WHGE	0.6			0.25		
Pagie1	R	3.42	338.66	4488.27	2440.7	400.88	2.16
	R/NL	3.32	114.39	1142.28	7975.03	579.07	0
	R/NL/NU	7.42	45.61	169.16	172.18	33.62	0
	GE	20.99			0		
	HGE	4.32			6.33		
	WHGE	2.75			6.86		
Text	R	6.5	65.12	176	10.5	3.93	0
	R/NL	7	88.06	176	154	25.23	0
	R/NL/NU	8.33	75.95	176	57	3.89	0
	GE	9.2			1.8		
	HGE	5.4			2.6		
	WHGE	5.4			3.2		

It can be seen that for each validation problem there is at least one evolved representation which is more effective than the GE baseline. On the other hand, all the human-designed baselines tend to perform better than the average evolved representation. It can also be seen that the R search strategy tends to be more effective than either R/NL or R/NL/NU: driving the evolution of the representation by redundancy only, thus, appears to be the more effective choice. It is interesting to note that the evolved representations tend to exhibit a much greater value for ΔBF than the baseline representations, that is, the evolved representations appear to be able to improve fitness during a search significantly.

The finding that the R strategy is more effective than either R/NL or R/NL/NU is confirmed also by Table 4. The table shows, for each problem and each of the three best representations obtained with each search variant, the average percentile rank of the final best fitness among all the validation runs (i.e., including other evolved representations) on that problem.

Finally, in order to gain further insights into the evolved representations, we analyzed the populations of the validation runs in terms of diversity at the level of phenotype and of fitness. We measured diversity as the rate of unique individuals

Table 4. Percentile ranks of three most effective representations for each search variant.

Search variant	n	KLand.-5	Pagie1	Text
R	1	0.003	0.086	0.003
	2	0.009	0.09	0.003
	3	0.316	0.021	0.003
R/NL	1	0.182	0.016	0.003
	2	0.059	0.252	0.035
	3	0.549	0.303	0.051
R/NL/NU	1	0.1	0.303	0.068
	2	0.311	0.303	0.103
	3	0.311	0.303	0.103

in the initial and in the final population. Table 5 shows the results, for each search variant and for each baseline: for the evolved representations, diversity were computed averaging across runs and across representations with the same search variant—e.g., the 0.66 initial phenotype diversity for R on the KLandscapes-5 problem is obtained by averaging the phenotype diversities measured at the first generation of the $5 \times 5 = 25$ validation runs performed with R search variant on that problem.

Table 5. Initial and final diversities.

		KLand.-5		Pagie1		Text	
	Search variant	In.	Fin.	In.	Fin.	In.	Fin.
Phenotype	R	0.66	0.5	0.99	0.49	0.93	0.21
	R/NL	0.95	0.68	1	0.27	0.68	0.19
	R/NL/NU	0.41	0.29	0.37	0.06	0.28	0.06
	GE	0.01	0	0.01	0	0.05	0
	HGE	0.49	0.31	0.58	0.01	0.95	0.03
	WHGE	0.45	0.46	0.63	0.06	0.92	0.02
Fitness	R	0.44	0.19	0.97	0.47	0.13	0.01
	R/NL	0.83	0.04	0.98	0.16	0.04	0
	R/NL/NU	0.35	0.01	0.37	0.05	0.03	0
	GE	0.01	0	0.01	0	0	0
	HGE	0.42	0	0.42	0	0.13	0
	WHGE	0.33	0.02	0.54	0.05	0.12	0

It can be seen that the populations evolved with the evolved representations are, in general, more diverse, than those evolved with the baselines, both from the point of view of the phenotype and of the fitness. However, we believe that this effect might be a result of the generally better search effectiveness of the baselines, which could lead to faster convergence of the population towards one or few (possibly locally) optimal solutions. On the other hand, it is interesting to note that the representations evolved with the R strategy appear more capable of preserving the population diversity: this finding confirms the interplay

existing between redundancy and diversity, which has already been highlighted in previous works [11].

6 Concluding remarks and future work

In the attempt of providing new insights into the long-standing problem of choosing the most appropriate representation for an EA, we have presented a method for the automatic synthesis of a representation for the large class of problems whose solutions spaces can be defined by a CFG. We have defined a representation template for genotype-phenotype mapping, in the form of a recursive function with two parameter functions that can be described using an ad hoc language that we have developed for this purpose. Our representation template is expressive enough to describe the classic GE mapping and more recent proposals such as HGE and WHGE; at the same time, our template is much more general and ensures that any instance representation is valid, i.e., it maps any input variable-length bit string to a string of the user-provided language in a finite number of steps. We used CFG-GP to evolve the representations expressed by our template with a multi-objective optimization of 3 crucial representation properties: redundancy, non-locality, and non-uniformity.

We executed a number of experiments and carefully assessed the evolved representations using human-designed representations proposed earlier in the literature, i.e., GE, HGE, and WHGE. The results show that our proposal indeed allows automatically designing a representation which exhibits better properties than the human-designed ones. However, only in few cases the evolved representations are also able to provide better search effectiveness. We hope that our work might open new research perspectives in the young field of automatic design of representations.

References

1. Altenberg, L.: 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 18(3), 363–367 (Sep 2017)
2. Castelli, M., Manzoni, L., Vanneschi, L., Silva, S., Popovič, A.: Self-tuning geometric semantic genetic programming. Genetic Programming and Evolvable Machines 17(1), 55–74 (2016)
3. Correia, M.B.: A study of redundancy and neutrality in evolutionary optimization. Evolutionary computation 21(3), 413–443 (2013)
4. Cruz-Salinas, A.F., Perdomo, J.G.: Self-adaptation of genetic operators through genetic programming techniques. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 913–920. GECCO ’17, ACM, New York, NY, USA (2017)
5. De Jong, K.: Parameter setting in eas: a 30 year perspective. Parameter setting in evolutionary algorithms pp. 1–18 (2007)

6. Fogel, D.B., Fogel, L.J., Atmar, J.W.: Meta-evolutionary programming. In: Signals, systems and computers, 1991. 1991 Conference record of the twenty-fifth asilomar conference on. pp. 540–545. IEEE (1991)
7. Foster, J.A.: Taking "biology" just seriously enough: 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 18(3), 395–398 (Sep 2017)
8. Hong, L., Drake, J.H., Woodward, J.R., Özcan, E.: A hyper-heuristic approach to automated generation of mutation operators for evolutionary programming. Applied Soft Computing (2017)
9. Lourenço, N., Pereira, F.B., Costa, E.: Sge: a structured representation for grammatical evolution. In: International Conference on Artificial Evolution (Evolution Artificielle). pp. 136–148. Springer (2015)
10. Mckay, R.I., Hoai, N.X., Whigham, P.A., Shan, Y., O'Neill, M.: Grammar-based genetic programming: a survey. Genetic Programming and Evolvable Machines 11(3-4), 365–396 (2010)
11. Medvet, E.: 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. p. to appear. Springer International Publishing, Cham (2017)
12. Medvet, E.: Hierarchical grammatical evolution. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO (2017)
13. Medvet, E., Bartoli, A., Squillero, G.: An effective diversity promotion mechanism in grammatical evolution. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. pp. 247–248. GECCO '17, ACM, New York, NY, USA (2017)
14. Medvet, E., Daolio, F., Tagliapietra, D.: Evolvability in grammatical evolution. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 977–984. GECCO '17, ACM, New York, NY, USA (2017)
15. Miller, J.F., Smith, S.L.: Redundancy and computational efficiency in cartesian genetic programming. IEEE Transactions on Evolutionary Computation 10(2), 167–174 (2006)
16. O'Neill, M., Brabazon, A., Nicolau, M., Garraghy, S.M., Keenan, P.: πGrammatical Evolution, pp. 617–629. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
17. O'Neill, M., Ryan, C.: Genetic code degeneracy: Implications for grammatical evolution and beyond. In: European Conference on Artificial Life. pp. 149–153. Springer (1999)
18. O'Neill, M., Ryan, C.: Grammatical evolution by grammatical evolution: The evolution of grammar and genetic code. Genetic Programming pp. 138–149 (2004)
19. Pagie, L., Hogeweg, P.: Evolutionary consequences of coevolving targets. Evolutionary computation 5(4), 401–418 (1997)
20. Pappa, G.L., Ochoa, G., Hyde, M.R., Freitas, A.A., Woodward, J., Swan, J.: Contrasting meta-learning and hyper-heuristic research: the role of evolutionary algorithms. Genetic Programming and Evolvable Machines 15(1), 3–35 (Mar 2014)
21. Qin, A.K., Huang, V.L., Suganthan, P.N.: Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE transactions on Evolutionary Computation 13(2), 398–417 (2009)
22. Rothlauf, F.: Representations for genetic and evolutionary algorithms. In: Representations for Genetic and Evolutionary Algorithms, pp. 9–32. Springer Berlin Heidelberg (2006)

23. Rothlauf, F., Goldberg, D.E.: Redundant representations in evolutionary computation. Evolutionary Computation 11(4), 381–415 (2003)
24. Ryan, C.: A rebuttal to whigham, dick, and maclaurin by one of the inventors of grammatical evolution: 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)
25. Ryan, C., Collins, J., Neill, M.O.: Grammatical evolution: Evolving programs for an arbitrary language, pp. 83–96. Springer Berlin Heidelberg, Berlin, Heidelberg (1998)
26. Saravanan, N., Fogel, D.B., Nelson, K.M.: A comparison of methods for self-adaptation in evolutionary algorithms. BioSystems 36(2), 157–166 (1995)
27. Scott, E.O., Bassett, J.K.: Learning genetic representations for classes of real-valued optimization problems. In: Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation. pp. 1075–1082. ACM (2015)
28. Simões, L.F., Izzo, D., Haasdijk, E., Eiben, A.E.: Self-adaptive genotype-phenotype maps: neural networks as a meta-representation. In: International Conference on Parallel Problem Solving from Nature. pp. 110–119. Springer (2014)
29. Spector, L.: Introduction to the peer commentary special section 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 18(3), 351–352 (Sep 2017)
30. Squillero, G., Tonda, A.: (over-)realism in evolutionary computation: 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. 1–3 (2017)
31. Thorhauer, A.: On the non-uniform redundancy in grammatical evolution. In: International Conference on Parallel Problem Solving from Nature. pp. 292–302. Springer (2016)
32. Thorhauer, A., Rothlauf, F.: On the locality of standard search operators in grammatical evolution. In: International Conference on Parallel Problem Solving from Nature. pp. 465–475. Springer (2014)
33. Vanneschi, L., Castelli, M., Manzoni, L.: The k landscapes: a tunably difficult benchmark for genetic programming. In: Proceedings of the 13th annual conference on Genetic and evolutionary computation. pp. 1467–1474. ACM (2011)
34. Whigham, P.A., Dick, G., Maclaurin, J.: On the mapping of genotype to phenotype in evolutionary algorithms. Genetic Programming and Evolvable Machines pp. 1–9 (2017)
35. Whigham, P.A., Dick, G., Maclaurin, J., Owen, C.A.: Examining the best of both worlds of grammatical evolution. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. pp. 1111–1118. ACM (2015)
36. Whigham, P.A., et al.: Grammatically-based genetic programming. In: Proceedings of the workshop on genetic programming: from theory to real-world applications. vol. 16, pp. 33–41 (1995)
37. White, D.R., Mcdermott, J., Castelli, M., Manzoni, L., Goldman, B.W., Kronberger, G., Jaśkowski, W., O'Reilly, U.M., Luke, S.: Better gp benchmarks: community survey results and proposals. Genetic Programming and Evolvable Machines 14(1), 3–29 (2013)
38. Wong, P.K., Wong, M.L., Leung, K.S.: Hierarchical knowledge in self-improving grammar-based genetic programming. In: International Conference on Parallel Problem Solving from Nature. pp. 270–280. Springer (2016)