Specializing Context-Free Grammars with a (1 + 1)-EA

Luca Manzoni, Alberto Bartoli, Mauro Castelli, Ivo Gonçalves, Eric Medvet

Luca Manzoni is with the Dipartimento di Matematica e Geoscienze, University of Trieste, 34127 Trieste, Italy. (email: lmanzoni@units.it)

Alberto Bartoli and Eric Medvet are with the Dipartimento di Ingegneria e Architettura (DIA), University of Trieste, 34127 Trieste, Italy. (email: bartoli.alberto@units.it and emedvet@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)

Ivo Gonçallves is with the INESC Coimbra, DEEC, University of Coimbra, Portugal. (email: icpg@dei.uc.pt)

Specializing Context-Free Grammars with a $(1+1)$-EA

Luca Manzoni, Alberto Bartoli, Mauro Castelli, Ivo Gonçalves, Eric Medvet

***Abstract*—Context-free grammars are useful tools for modeling the solution space of problems that can be solved by optimization algorithms. For a given solution space, there exists an infinite number of grammars defining that space, and there are clues that changing the grammar may impact the effectiveness of the optimization. In this paper, we investigate theoretically and experimentally the possibility of *specializing* a grammar in a problem, that is, of systematically improving the quality of the grammar for the given problem. To this end, we define the quality of a grammar for a problem in terms of the average fitness of the candidate solutions generated using that grammar. Theoretically, we demonstrate the following findings: (a) that a simple mutation operator employed in a $(1+1)$-EA setting can be used to specialize a grammar in a problem without changing the solution space defined by the grammar; and (b) that three grammars of equal quality for a grammar-based version of the OneMax problem greatly vary in how they can be specialized with that $(1+1)$-EA, as the expected time required to obtain the same improvement in quality can vary exponentially among grammars. Then, experimentally, we validate the theoretical findings and extend them to other problems, grammars, and a more general version of the mutation operator.**

***Index Terms*—Grammar Design, Run Time Analysis, Grammatical Evolution.**

I. Introduction

Context-free grammars (CFGs) [1] are powerful and widespread tools for describing languages in a concise way. Several evolutionary algorithms (EAs) use CFGs for describing the space of solutions [2, 3, 4, 5] on the assumption that any string of the language described by the CFG is a syntactically valid candidate solution to the problem at hand. In such frameworks, the evolutionary search is guaranteed to produce only valid candidate solutions without the need to design and execute additional problem-specific checks.

Because a given language can be described by an infinite number of CFGs, a solution space that can be defined in the form of a language may result from infinitely many CFGs as well. This fact is important from an evolutionary computation point of view because the specific CFG used affects the *representation* of the candidate solutions. The degree to which the CFG determines the representation varies among grammar-based EAs, in general being smaller in *indirect* representations (as in grammatical evolution [3]) and larger in *direct* representations (as in CFG genetic programming [6]). It is well-known, however, that representation is a key component of every EA in determining the *structure* of the solution space in terms of how easily the search algorithm can move from one candidate solution to another by applying genetic operators [7, 8, 9].

It follows that, for a given grammar-based EA, a given fitness function, and a given solution space, there may be very different *fitness landscapes* [10, 11] depending on the actual CFG used to describe the solution space. In other words, the CFG plays an important role in determining the degree of hardness of the overall optimization scenario.

The relationship between CFGs and fitness landscapes might be exploited to carefully shape the fitness landscape and make the specific problem at hand easier to solve [12]. For example, a CFG could be restructured to repeat the production rules for symbols that should occur more often in candidate solutions of good quality [13, 14]. However, beyond such proposals, there is no established practice for tailoring a CFG to make it more suitable for an EA or a problem. Indeed, the few experimental pieces of evidence that the CFG might impact the EA's effectiveness (with the same language, fitness function, and EA) do not explain such a dependency [15, 16].

In this work, we focus on the role of the CFGs in EAs and, in particular, on the quality of a CFG for a given problem in terms of the average fitness of the candidate solutions generated using that CFG. We investigate whether we can *evolve* an existing CFG in order to improve its quality for the given problem, that is, whether we can *specialize* a CFG in that problem. Our goal is to provide a framework with theoretical foundation and experimental validation for reasoning about the following questions: can CFGs be specialized in a problem? Are different CFGs equally suitable to be specialized in a problem? Is a CFG equally suitable to be specialized in different problems?

The intuition behind specialization is that given two CFGs G_1 and G_2 generating the same language, if the probability of randomly generating a solution near the optimal solution is higher in G_1 than in G_2, then we might expect usage of G_1 to be more advantageous than usage of G_2 for the EA itself. In other words, each CFG embodies a certain set of biases in the generation of the candidate solutions that are reflected in the resulting fitness landscape; our crucial research question is whether CFGs can be evolved to have positive biases in this respect. This way, the user would provide a CFG capable of constructing the set of possible candidate solutions, and such a

Luca Manzoni is with the Dipartimento di Matematica e Geoscienze, University of Trieste, 34127 Trieste, Italy. (email: lmanzoni@units.it)
Alberto Bartoli and Eric Medvet are with the Dipartimento di Ingegneria e Architettura (DIA), University of Trieste, 34127 Trieste, Italy. (email: bartoli.alberto@units.it and emedvet@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)
Ivo Gonçalves is with the INESC Coimbra, DEEC, University of Coimbra, Portugal. (email: icpg@dei.uc.pt)
Manuscript received . . .

CFG would then be refined automatically to improve its ability to construct good solutions. Ideally, a complex evolutionary search might even be unnecessary; if a CFG is able to generate the global optimum with high probability, then a short random search performed by generating solutions according to the CFG biases might be sufficient. We remark, however, that we do not investigate the practicality of CFG specialization. From a computational point of view, an evolutionary search on the unspecialized CFG might turn out to be more efficient than either a short random search on the specialized CFG or a full evolutionary search on a partially specialized CFG. Our work does not address these important questions and focuses instead on ascertaining whether CFG specialization is actually possible in the first place.

Our approach is based on two central ideas: the definition of a *mutation operator for CFGs* (i.e., an operator that modifies a CFG into another that defines the same language) and that of a synthetic proxy measure for the *"quality" of the fitness landscape*, which we define based on the expected fitness of a solution generated by a given CFG. Although this measure clearly cannot capture the full complexity of a fitness landscape, it has the advantage of being tractable from a theoretical point of view. For example, we will be able to determine how this measure changes during an evolutionary search in the CFG space.

For the purpose of conducting a theoretical analysis of the specialization, we define a CFG-based problem that mimics the ONEMAX problem for traditional binary strings, consider a particular case of the CFG mutation operator that corresponds to changing the probability of selecting production rules, and apply this operator in a $(1+1)$-EA on CFGs—we choose this EA because it is commonly employed in initial theoretical analyses. We use this framework to prove that different CFGs for the same language of the ONEMAX problem indeed are of different qualities, and that CFGs with the same initial quality take very different times to be evolved to reach a similar quality (i.e., CFGs of the same quality are differently suitable to be specialized).

We then validate and extend these theoretical findings by performing a suite of experiments on several other problems resulting from different combinations of language and fitness function; we use an estimate of the quality of a CFG, i.e., of the expected fitness of the generated solutions, instead of the actual value. For each language, we consider many CFGs with the same initial quality and show that the theoretical findings for the ONEMAX problem are qualitatively confirmed. We also show that which CFG is the most suitable to be specialized depends on the problem: the fitness function may favor some structures in the solutions and different CFGs are more or less prone to be specialized for expressing those structures.

We believe that our findings mark a first step towards understanding the dependency between the efficiency of an EA and the quality of the CFG defining the search space. More broadly, we hope that our study can foster the research in CFG specialization—when and how to do it in practice—and CFG design—how to design a CFG to favor its specialization.

II. RELATED WORKS

We are not aware of any previous study that formally addresses the questions considered in the present paper—that is, if and how a CFG can be specialized in a problem. There are, however, several studies that investigate similar issues or, indirectly, concern the modification of a CFG. We here review the most significant ones.

A. Solution structure and model sampling

The topic studied in this paper belongs to a research stream that dates back to the 1990s, when different approaches aimed at explicitly characterizing the structure of good candidate solutions in genetic programming (GP) [17, 18] began to appear. An important finding in this respect is the fact that GP search effectiveness can improve if potentially useful sub-solutions can be identified and used as building blocks in the evolutionary process. Motivated by these findings, researchers proposed specific crossover operators for attempting to preserve such building blocks and taking advantage of them [19, 20]. It was later observed, though, that commonly used genetic operators (crossover and mutation) tend to generate new solutions by destroying useful building blocks instead of taking advantage of them [21].

For overcoming such limitation one can build a model of fitter solutions and then sample this model to generate new solutions—that is, without using genetic operators. Several approaches based on this idea have been proposed in several evolutionary computation frameworks. In the context of genetic algorithms (GAs), estimation of distribution algorithms (EDAs) [22] and their variants [23, 24] construct a probabilistic model of good-quality solutions over the space of all possible solutions. The model may be refined during the search by focusing on the information that may be extracted from fitter solutions. Learnable evolution model (LEM) [25] builds inductive hypotheses for explaining why certain solutions outperform others at the given task and then uses those hypotheses for generating new solutions. In GP, probabilistic incremental program evolution (PIPE) [26] combines probability vector coding of program instructions, population-based incremental learning [27], and tree-coded programs. Estimation of Distribution Programming (EDP) [28] models dependencies among nodes in a tree representation with a Bayesian network.

The research stream closer to the work presented in this paper is the one of grammar-model-based techniques. Whigham described the use of a stochastic CFG to define the structure of the initial language [2, 29]. The success of solutions gradually biased the probabilities of productions, with an evolutionary search that still used crossover and mutations. Biasing the probabilities of productions attempts to shape the search space while the evolution proceeds, with grammars that become more specialized to the problem at hand. The effect of different sources of bias is further studied by Whigham [30]. A stochastic grammar-based GP (SG-GP) was proposed by Ratle and Sebag [31]. SG-GP uses a CFG and can be considered the first pure grammar-based EDA-GP system [21]; at each generation, a new population is created from the current probability distribution and the distribution is subsequently updated taking into account solutions in the current population. A similar idea was exploited by Abbass et al. [32], who used ant colony optimization [33] to construct the new population of solutions.

B. Grammar learning

The aforementioned studies only modify the probabilities associated with the different production rules, not the grammar structure. As discussed by McKay et al. [21], this property creates a dilemma: an initial grammar that is excessively simple could make the task impossible to solve (i.e., because it represents wrong dependencies, or because it does not represent important dependencies in the solution), while an initial grammar that is too complex could lead to a complex parameter learning problem (thereby requiring a considerable amount of time to converge).

Based on these considerations, more recent studies advocated using some form of grammar learning. This line of research includes program evolution with explicit learning (PEEL) [34], the method proposed by Bosman and de Jong [35], and the grammar model-based program evolution (GMPE) [36]. The former uses a search space description table (SSDT) to describe the search space and employs ant-colony optimization to perform grammar refinement. The search starts from a minimal SSDT providing a high-level description of the search space, while the optimization updates the stochastic components of the SSDT. When the SSDT is deemed inadequate to focus the search on more promising areas of the search space, the description table is modified by splitting certain rules in the grammar. In contrast to the other methods, genotypes are represented purely as expression trees, not as derivation trees. Thus, when a new grammar is learned from the selected population, the solutions must first be parsed. GMPE is similar to the cited work, the major difference being that GMPE learns from the intermediate genotype while the method in [35] learns directly from the expression trees. A broader review of probabilistic model building in GP may be found in [37].

C. Genotype-phenotype mapping

A different perspective on the quality of solutions resulting from an evolutionary search was taken by works that highlighted the importance of the genotype-phenotype mapping for determining the exploration strategy of an evolutionary search. Toussaint [38] analyzed how multiple genotypes representing the same phenotype might pose different ways of exploring the search space. Authors define this phenomenon as self-adaptability. By use of a system that encodes the phenotype with a grammar-like genotype, a large variability of exploration strategies for a fixed phenotype and a self-adaptive drift towards short representations were observed. Palacios et al. [39] considered a neurogenetic model with a genotype-phenotype map of a highly degenerate nature. Despite the nature of the mapping, certain genotypes were consistently preferred, thus indicating that the genotype-phenotype symmetry was broken. The authors explained that while degenerate genotypes were equivalent in terms of reproductive selection, the genetic operators broke the symmetry, picking out the more robust or less brittle genotypes (i.e., those that were most likely to lead to other fitter candidate solutions).

The importance of the mapping to the exploration strategy was made even more evident by Wilson and Kaur [40], who showed how random mutations on genes make non-random phenotype preferences, based on the structure of a map. In particular, the interaction between such mutation-based preferences and fitness preferences can explain population dynamics on neutral landscapes. Additionally, the authors specified conditions under which increasing degeneracy or rearranging the rules of a grammar do not affect performance. The impact of redundant genotype-to-phenotype mapping under Boolean linear GP was studied by Nickerson et al. [41]. Their analysis was performed with a network that represents the possible transitions from genotypes to phenotypes. The concepts of robustness, evolvability, and accessibility of phenotypes are analyzed. Results show that more robust phenotypes are both more evolvable and more accessible.

D. Grammatical evolution

The effect of grammar design on search outcomes has been studied considerably in the grammatical evolution (GE) framework. The co-evolution of grammar and genetic code has been studied by O'Neill and Ryan [42]. This approach uses two grammars—the universal and the solution grammar, the former specifying how to construct the latter. Since the rules described in the universal grammar guide the mapping of each individual, these rules can be evolved to allow specific bias for certain symbols or to dynamically reduce the search space. Experiments on a set of symbolic regression problems show that the approach is able to dynamically bias the search process toward the terminal symbols more relevant for the problem being considered. The approach was elaborated further in [43], which explored a meta-grammar GA (mGGA) and compared variants with static and dynamic grammars. Dynamic grammars outperformed static grammars on all the experiments considered (several configurations of ONEMAX, ZEROMAX, and a deceptive trap problem based on Trap5). Hemberg et al. [44] provided an analysis of an mGGA focusing on the building block structures and the grammar design. Results showed that the building block structures were adapted successfully and that grammar design does play an important factor in terms of search efficiency.

Nicolau [45] proposed a systematic procedure for reducing the number of non-terminal symbols in a grammar, along with an extensive experimental assessment of the impact such a procedure might have on search effectiveness. The study was inconclusive, as no statistically significant differences were found between using the original grammar and using the corresponding simplified grammar. More consistent results regarding the relation between grammar structure and search effectiveness have been found recently by Nicolau and Agapitos [12] in the context of GE with linear genome representation. The cited work mostly focused on symbolic regression problems and considered several grammar design aspects, including grammar balancing, type of notation, and symbol biases. The results showed that different grammar designs can indeed influence the search effectiveness significantly, with the most prominent and consistent performance improvements being associated with the usage of recursion-balanced grammars and the reduction of non terminals.

The relation between grammars and search behavior was also studied for very specific aspects. Hemberg et al. [46] examined the effect of postfix, prefix, and infix grammar in symbolic regression problems. The key results were that postfix grammars tend to deliver a performance advantage in harder problems and that prefix grammars tend to generate a large amount of invalid individuals—that is, genotypes that cannot be transformed to any phenotype by the mapping procedure.

III. Theoretical analysis

A CFG, simply *grammar* in the following, is a tuple $G = (N, T, s_0, R)$ that concisely defines a language $L(G)$ over the alphabet T. In the tuple G, N is the set of non-terminal symbols, T is the set of terminal symbols (with $T \cap N = \emptyset$) or alphabet, $s_0 \in N$ is the starting symbol or axiom, and R is the set of production rules. A production rule consists of a non-terminal on the left-hand side (LHS) and a sequence of terminals and non-terminals on the right-hand side (RHS).

In the next sections, we will define the concept of the quality of a grammar G in the context of an optimization problem where the search space is the language $L(G)$. We will then describe a class of modifications to a grammar that do not alter the language defined by it but might alter the grammar's quality. After that description we will introduce an EA based on the aforementioned modifications that enables specialization of a grammar. Finally, we will consider a case in which the optimization problem is OneMax, and we will derive upper and lower bounds for the expected quality obtained with the EA when applied to three different base grammars, all describing the search space of OneMax and all with the same quality. Hence, we will show that the degree to which different grammars can be specialized by acting on their rules greatly differs, even when they define the same language and have the same initial quality.

In Section III-G, we will extend part of the theoretical findings to the larger and more interesting case of infinite languages, reasoning on an infinite version of the OneMax problem.

Later, in Section IV, we will experimentally validate these theoretical findings and extend them to the following: (a) a class of intermediate grammars that exhibit, at the same time, traits of two of the base grammars, hence being more representative of real grammars; (b) different problems with respect to OneMax resulting from combinations of different fitness functions and languages; and (c) a more general version of the grammar modification.

A. Probability function of a grammar

We say that a string s can be *generated* using a grammar G if it can be obtained as the result of the following procedure. Initially, s is set to the axiom of the grammar. Then, until s does not contain any non-terminals, the following steps are iterated: (i) a non-terminal $\alpha \in N$ is chosen in s; (ii) one production rule r is chosen with uniform probability $\frac{1}{|R_\alpha|}$ among the rules R_α whose LHS is α; and (iii) α is replaced in s with the RHS of the chosen rule r. As an aside, it can be noted that this generation procedure resembles the one employed by many grammar-based approaches (e.g., CFG genetic programming [2], structured grammatical evolution [4], and weighted grammatical evolution [5]) in which the production rule is chosen with uniform probability.

The set of strings that can be generated using G with this procedure corresponds to the language $L(G)$ defined by G. That is, every string in the language can be generated with a positive (even if arbitrarily small) probability. We describe these probabilities using a probability function:

Definition 1. *Given a grammar G defining the language $L(G)$, the* probability function of G *is a function $p_G : L(G) \to (0, 1]$ such that $p_G(s)$ is the probability of generating the string $s \in L(G)$ using G.*

It holds that $\sum_{s \in L(G)} p_G(s) = 1$. By definition, because for any $s \in L(G)$ $p_G(s) > 0$, it means that the set of strings that can be generated using G is exactly $L(G)$.

B. Quality of the grammar for a problem

When a grammar describes the search space $L(G)$ of an optimization problem in which the fitness is a function $f : L(G) \to \mathbb{R}$ to be maximized, the probability function of G impacts the probability of generating candidate solutions with good fitness.

We define the *quality* of a grammar G for a problem defined by $L(G)$ and f as the expected fitness of a string generated by G:

Definition 2. *Given a grammar G and a fitness function $f : L(G) \to \mathbb{R}$, the* quality of G *is defined as:*

$$q_f(G) = \sum_{s \in L(G)} p_G(s) f(s) \tag{1}$$

That is, on average, the grammar G will generate a solution of fitness $q_f(G)$. From another point of view, if a grammar G_1 exists such that $L(G_1) = L(G_2)$ and $q_f(G_1) > q_f(G_2)$, this means that G_1 will generate, on average, solutions with better fitness than the ones generated by G_2.

Let $s^\star \in L(G)$ be an optimal solution—that is, one for which $\forall s \in L(G), f(s) \leq f(s^\star)$. Furthermore, let us assume that there exists $s^\dagger \in L(G)$ such that $f(s^\dagger) < f(s^\star)$—that is, that not all solutions have the same (optimal) fitness. It can be seen that it is impossible to have $q_f(G) = f(s^\star)$ because this would imply that for all strings $s \in L(G)$ with $f(s) < f(s^\star)$, the probability to be generated is equal to 0 and, by hypothesis, at least one such string $s^\dagger$, exists. However, $q_f(G)$ can come arbitrarily close to the optimal value $f(s^\star)$.

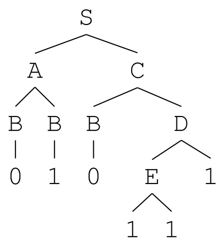
The quality of the grammar, as defined here, is a proxy measure for the quality of the fitness landscape that arises when an EA is used to solve the problem defined by $L(G)$ and f. The key idea is that if a better candidate solution has a higher probability of being generated, then the entire evolutionary search can reach the optimum faster. In fact, it has already been shown experimentally that the quality of the initial population [15] and the biases in the grammar [12] can greatly influence the effectiveness and efficiency of the search. However, we remark that degenerate cases exist: for example, with trap functions [47], increasing the average fitness of the generated candidate solutions might be counterproductive because we might favor the convergence toward local optima, rather than the global optimum.

C. Shortcut grammar mutation

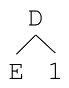
Here, we define a class of modifications of a grammar G into a grammar G' that do not alter the generated language $L(G) = L(G')$, but might result in a different probability function $p_G \neq p_{G'}$.

The modifications are defined based on the concept of the derivation tree, an intermediate outcome of the process of generating a string described in Section III-A. The *derivation tree* of a string $s \in L(G)$ generated with G is a tree for which the root is the axiom, leaf nodes are terminal symbols, non-leaf nodes are non-terminal symbols, and the children of a node are the symbols corresponding to the RHS of the rule used during the generation of s for replacing the node.

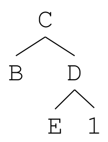
Let us consider the following derivation tree, derived from some grammar G with $T = \{\mathtt{0}, \mathtt{1}\}$ and $N = \{\mathtt{S}, \mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}\}$:



Now, let us focus on a single internal node of the derivation tree and take it as a root of a tree of height 1 representing a subset of the original derivation tree. For example, if D is selected as an internal node, the resulting tree will be:



The entire connected subset of the derivation tree can be encoded into a single production rule in the following, where the LHS is the non-terminal symbol at the root of the tree and the RHS is obtained by concatenating the leaves of the tree from left to right. In this example, the resulting rule is $\mathtt{D} \rightarrow \mathtt{E1}$. From the definition of the derivation tree, it follows that this rule corresponds to an already existing rule in the grammar: this is always true when we consider subsets of the derivation tree of height 1. It is not true, in general, when trees of greater height are considered. For example, by repeating the same procedure with the following connected subset of the derivation tree:



the resulting rule is $\mathtt{C} \rightarrow \mathtt{BE1}$, which is not necessarily a rule in the considered grammar G (i.e., it might or might not already exist: this cannot be inferred by looking only at one single derivation tree). While the addition of the rule $\mathtt{C} \rightarrow \mathtt{BE1}$ does *not* change the language generated by the grammar, the exact rule might not be already present among the production rules of the grammar.

We generalize this process and hence define the *shortcut mutation of a grammar G into a grammar G'* as follows:

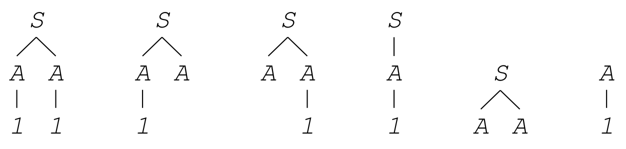
1) a string is generated using G and its derivation tree is considered;
2) a connected subset τ of the derivation tree is selected starting from a node α randomly selected from a uniform distribution such that, for each of the descendants of α, either (a) all of its children in the derivation tree are included in the set τ (b) or none of its children are. The subset τ is itself a tree rooted in α.
3) a new rule is built where the LHS is the node α and the RHS is the concatenation of the leaf nodes of the tree τ enumerated from left to right (i.e., preserving their ordering);
4) G' is built from G by adding the newly defined rule.

One of the conditions 2a and 2b is required to hold for each node because taking only one or more but not all children of a node would correspond to a partially applied production rule, a situation that would require additional care to avoid changing the language generated by the grammar, as highlighted in the following example.

Example 1. *Let us consider the following grammar G:*

$$S \rightarrow AA$$
$$A \rightarrow 1$$

which has only one possible derivation tree. The previously defined rules for shortcut mutation ensure that among the following distinct connected subsets of the derivation tree, all but the fourth one can be considered for shortcut mutation.



They define the rules $S \rightarrow 11$, $S \rightarrow 1A$, $S \rightarrow A1$, $S \rightarrow 1$, $S \rightarrow AA$, and $A \rightarrow 1$. Among those, only the rule $S \rightarrow 1$ does not preserve the language generated by G, adding to it the string 1.

It can be seen that the shortcut mutation does not alter the generated language and might modify the probability function. The latter (i.e., $p_G \neq p_{G'}$) has, as a necessary condition, the addition of a new way of deriving at least one string in $L(G)$. This condition alone, however, is not sufficient: for example, in a grammar generating only a single string s, adding additional derivations for s does not change the probability function of the grammar.

Consider, for example, a trivial case in which $\mathtt{A} \rightarrow \mathtt{0}$ and $\mathtt{A} \rightarrow \mathtt{1}$ are the only rules in a grammar G defining the language

$\{0,1\}$. In this case, $p_G(0) = p_G(1) = \frac{1}{2}$. By applying the shortcut mutation to G, the first rule could be duplicated, and hence the probability function could be modified such that $p_G(0) = \frac{2}{3}$ and $p_G(1) = \frac{1}{3}$. Alternatively, the second rule could be duplicated, resulting in the opposite situation. Thus, the mutation has an effect on the quality of the grammar: some solutions will be generated with a higher probability than others.

It is interesting to point out that the shortcut mutation where the selected subset forms a tree of height limited to 1 corresponds to duplicating existing rules in the grammar. This technique is practically relevant and has been used in GE to change the bias of a grammar [12], often based on some knowledge about the problem being tackled (i.e., about G and f).

D. Specializing a grammar with a $(1+1)$-EA

Here, we show how we can tackle the problem of specializing a grammar in a problem with a (1+1)-EA based on the shortcut mutation operator defined in the previous section. We choose a $(1+1)$-EA, as it facilitates the aforementioned theoretical analysis.

Algorithm 1 shows the simple $(1+1)$-EA that, by iteratively applying the shortcut mutation, can specialize a grammar. In this EA, individuals (i.e., grammars) are compared according to their quality—that is, the fitness of an individual G is its quality $q_f(G)$.

```
Input: A grammar $G$ defining the language $L(G)$; a
       fitness function $f : L(G) \to \mathbb{R}$; a finite number
       of iterations $l$.
Output: A grammar $G'$ such that $L(G) = L(G')$ and
        $q_f(G') \geq q_f(G)$.
1 foreach $i \in \{1, \dots, l\}$ do
2     $G' \leftarrow$ Mutate($G$)
3     if $q_f(G') \geq q_f(G)$ then
4         $G \leftarrow G'$
5     end
6 end
```
Algorithm 1: $(1+1)$-EA for specializing a grammar.

By reasoning on the expected fitness of this EA as a function of the starting grammar G, we can reason on the degree to which G can be specialized.

It is important to notice that, in general, the quality of a grammar cannot be computed efficiently, or even exactly, because it involves knowing the fitness and the probability of all and every possible string in the language. For this reason, we will consider a particular set of grammars, each generating the entire set of binary strings of a fixed length n, and in which the fitness is given by the OneMax problem (i.e., the fitness is equal to the number of bits set to one). This will allow us to derive upper and lower bounds for the expected fitness of the EA of Algorithm 1 applied to different grammars for the OneMax problem that have the same initial quality.

$$\begin{aligned} \mathtt{A}_1 &\to \mathtt{0A}_2 \mid \mathtt{1A}_2 \\ &\dots \\ \mathtt{A}_{n-1} &\to \mathtt{0A}_n \mid \mathtt{1A}_n \\ \mathtt{A}_n &\to \mathtt{0} \mid \mathtt{1} \end{aligned}$$

(a) Grammar G_a.

$$\begin{aligned} \mathtt{B}_1 &\to \mathtt{B}_2\mathtt{B}_2 \\ &\dots \\ \mathtt{B}_m &\to \mathtt{B}_{m+1}\mathtt{B}_{m+1} \\ \mathtt{B}_{m+1} &\to \mathtt{0} \mid \mathtt{1} \end{aligned}$$

(b) Grammar G_b.

$$\mathtt{C} \to \mathtt{0}^n \mid \mathtt{0}^{n-1}\mathtt{1} \mid \dots \mid \mathtt{1}^{n-1}\mathtt{0} \mid \mathtt{1}^n$$

(c) Grammar G_c.

Fig. 1: Grammars G_a, G_b, and G_c for the binary strings of length n.

E. Three grammars for the OneMax problem

The prototypical problem for the study of GAs and, more generally EAs, is the OneMax problem [48, 47]. In the OneMax problem, the search space is the set of binary strings of length n, and the fitness of a string s is the number $f(s) = |s|_1$ of bits set to one, to be maximized.

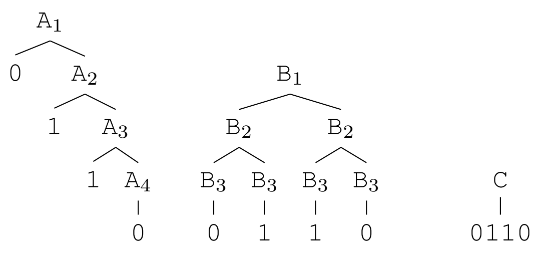
The OneMax problem has a unique maximum at $s = 1^n$, the string of all ones. Depending on the EA that is used, OneMax might also have no local optima. These characteristics make OneMax an ideal problem to be studied from a theoretical point of view. A previous result of particular interest with respect to our study concerns the expected number of iterations for reaching the global optimum in OneMax with a $(1+1)$-EA employing a bit flip mutation. That number is $\mathcal{O}(n \log n)$, which corresponds to the expected number of iterations needed to flip all bits of an n-bits string at least once [48, 47].

Because here we deal with grammars, we consider the grammars that define the language of binary strings of length n (i.e., such that $L(G) = \{0,1\}^n$). We acknowledge that this is a very restricted case because this language is finite and, in most problems of practical interest, the corresponding languages (and search spaces) are infinite. However, for the sake of clarity and tractability, we start by considering this case—we will extend part of the results to the case of infinite languages in Section III-G.

We consider the three different grammars G_a, G_b, and G_c shown in Figure 1 using the Backus-Naur form. Without loss of generality, we define G_b only for n being a power of two ($m = \log_2 n$ in Figure 1b). Grammar G_a has $2n$ rules, G_b has $1 + \log_2 n$ rules, and G_c has 2^n rules.

Figure 2 shows three examples of derivation trees obtained with G_a, G_b, and G_c and with $n = 4$. From the respective grammars, it can be deduced that all the derivation trees generated with these grammars and this n will have the same shapes as the trees in Figure 2.

It can be seen that the probability function p for the three grammars is the same and corresponds to the uniform probability—that is, every string in $L(G_a) = L(G_b) = L(G_c) = \{0,1\}^n = L_n$ can be generated with the same



(a) Derivation tree with G_a. (b) Derivation tree with G_b. (c) Derivation tree with G_c.

Fig. 2: Example derivation trees obtained with the grammars G_a, G_b, and G_c of Figure 1 and $n = 4$.

probability:

$$p(s) = \frac{1}{2^n} \tag{2}$$

The quality $q_f(G_a) = q_f(G_b) = q_f(G_c) = q_n$ of the grammars for OneMax is hence:

$$q_n = \sum_{s \in L_n} p(s)|s|_1 = \sum_{s \in L_n} \frac{1}{2^n}|s|_1 = \frac{1}{2^n}\sum_{s \in L_n}|s|_1 \tag{3}$$

which means that the quality of the grammar can be computed as the average number of ones into the strings of the language. By enumerating all the 2^n distinct strings of the language, one can observe that, on average, a string has $\frac{n}{2}$ positions equal to one, thus showing that:

$$q_n = \frac{1}{2^n}\sum_{s \in L_n}\frac{n}{2} = \frac{n}{2} \tag{4}$$

This is a sound yet unsurprising result; the key point, however, is that these three grammars define the same language and have the same quality. In the next section, we will show that, in spite of this premise, the outcomes of applying the $(1+1)$-EA of Algorithm 1 to each of the three grammars are different.

F. Bounds for the $(1+1)$-EA on G_a, G_b, and G_c

Here, we derive the upper (for G_a and G_c) and lower (for G_a and G_b) bounds for the expected fitness of the EA of Algorithm 1, that is, the expected quality of the grammar after a given number of iterations of the EA.

We restrict the analysis to the case in which only shortcut mutations of height 1 are applied in the EA: as discussed before, this corresponds to adding multiple copies of a single production rule in the grammar. This restriction makes the derivation of the bounds easier and the result more comprehensible. In Section IV, we experimentally extend the study to the case of heights larger than 1. This case is of particular interest because applying the shortcut mutation with a height greater than 1 in the context of the $(1+1)$-EA corresponds essentially to the search for useful building blocks. The goal of finding useful building blocks has been pursued by different researchers in the past, on varying EAs and with different approaches (e.g., in [49, 50] for grammar-based genetic programming).

1) Grammar G_a: We start by considering the grammar G_a. Let us consider the expected fitness obtained by the $(1+1)$-EA after $\mathcal{O}(n \log n)$ iterations. We recall that $\mathcal{O}(n \log n)$ is the *expected* number of iterations needed to flip all bits in the OneMax problem with $(1+1)$-EA employing the bit flip mutation. Similarly, the *expected* number of iterations after which at least one rule of the forms $\mathtt{A}_i \rightarrow \mathtt{1A}_{i+1}$ or $\mathtt{A}_n \rightarrow \mathtt{1}$ has been added for each non-terminal $\mathtt{A}_1, \dots, \mathtt{A}_n$ of G_a is $\mathcal{O}(n \log n)$. Notice that production rules of the forms $\mathtt{A}_i \rightarrow \mathtt{0A}_{i+1}$ or $\mathtt{A}_n \rightarrow \mathtt{0}$ can never be accepted by the EA because the resulting grammar will have a lower quality, as strings with $\mathtt{0}$ in the ith bit, for any value of i, would have an increased probability of being generated. Because each production rule of the form $\mathtt{A}_i \rightarrow \mathtt{1A}_{i+1}$ or $\mathtt{A}_n \rightarrow \mathtt{1}$ has been duplicated, in expectation, at least once, we can assume that each $\mathtt{1}$ in the string has an expected probability of at least $\frac{2}{3}$ of being generated (and, thus, that each $\mathtt{0}$ has an expected probability of at most $\frac{1}{3}$ of being generated). This means that the expected fitness of the resulting grammar G will be at least:

$$q_f(G) = \sum_{s \in L_n} p(s)|s|_1 \geq \sum_{s \in L_n}\left(\frac{1}{3}\right)^{|s|_0}\left(\frac{2}{3}\right)^{|s|_1}|s|_1 \tag{5}$$

We can perform the following simplifications:

$$\begin{aligned}
q_f(G) &\geq \sum_{s \in L_n}\left(\frac{1}{3}\right)^{|s|_0}\left(\frac{2}{3}\right)^{|s|_1}|s|_1 \\
&= \sum_{s \in L_n}\left(\frac{1}{3}\right)^{n-|s|_1}\left(\frac{2}{3}\right)^{|s|_1}|s|_1 \\
&= \sum_{k=0}^{k=n}\binom{n}{k}\left(\frac{1}{3}\right)^{n-k}\left(\frac{2}{3}\right)^{k}k \\
&= 3^{-n}\sum_{k=0}^{k=n}k2^k\binom{n}{k} = 3^{-n}3^{n-1}2n = \frac{2}{3}n
\end{aligned} \tag{6}$$

In general, for every $\ell \in \mathbb{N}^+$, we can compute a bound on the expected value of the fitness after $\mathcal{O}(\ell n \log n)$ iterations by performing the same calculations as before:

$$\begin{aligned}
q_f(G) &= \sum_{s \in L_n} p(s)|s|_1 \\
&\geq \sum_{s \in L_n}\left(\frac{1}{\ell}\right)^{|s|_0}\left(\frac{\ell-1}{\ell}\right)^{|s|_1}|s|_1 \\
&= \sum_{s \in L_n}\left(\frac{1}{\ell}\right)^{n-|s|_1}\left(\frac{\ell-1}{\ell}\right)^{|s|_1}|s|_1 \\
&= \sum_{k=0}^{k=n}\binom{n}{k}\left(\frac{1}{\ell}\right)^{n-k}\left(\frac{\ell-1}{\ell}\right)^{k}k \\
&= \ell^{-n}\sum_{k=0}^{k=n}k(\ell-1)^k\binom{n}{k} \\
&= \ell^{-n}\ell^{n-1}(\ell-1)n \\
&= \frac{\ell-1}{\ell}n
\end{aligned} \tag{7}$$

which, when ℓ increases, converges (in the limit) to n. We can then state the following theorem:

Theorem 1. *For each $\ell \in \mathbb{N}^+$, starting from the grammar G_a defining the language L_n, the grammar generated by the $(1+1)$-EA employing the shortcut mutation of height 1 will have an expected quality after $\mathcal{O}(\ell n \log n)$ iterations greater than or equal to $\frac{\ell-1}{\ell}n$.* □

Notice that we are not taking into account the fact that mutations up to a given iteration result in a grammar with greater quality and hence with greater probability of generating strings with more ones than zeros: this increases the chance of selecting the corresponding production rules containing ones with the shortcut mutation. In turn, this means that the lower bound posed by Theorem 1 is particularly conservative.

To establish an upper bound for the quality of the grammar, we consider the expected quality after ℓn iterations. In fact, n is the number of iterations required, in the best case, to duplicate at least once each rule of the form $\mathtt{A}_i \rightarrow \mathtt{1A}_{i+1}$ and $\mathtt{A}_n \rightarrow \mathtt{1}$; similarly to the case of the lower bound, ℓn iterations are required to add ℓ copies of each of those rules—that is, the same outcome of the worst case is here obtained in ℓn steps instead of $\mathcal{O}(\ell n \log n)$ iterations. In this case, by convexity of the binomial function, the best situation is when mutations are uniformly distributed among all the rules of the form $\mathtt{A}_i \rightarrow \mathtt{1A}_{i+1}$ and $\mathtt{A}_n \rightarrow \mathtt{1}$. Therefore, we obtain the following result:

Theorem 2. *For each $\ell \in \mathbb{N}^+$, starting from the grammar G_a defining the language L_n, the grammar generated by the $(1+1)$-EA employing the shortcut mutation of height 1 will have an expected quality after ℓn iterations lower than or equal to $\frac{\ell-1}{\ell}n$.* □

This theorem shows that performing random mutations is, in this particular case, slower by a logarithmic factor with respect to the best case.

2) Grammar G_b: We now consider the second grammar, G_b. By observing Figure 1b and Figure 2b, it can be noted that selecting a random node inside the derivation tree that is *not* labeled with $\mathtt{B}_{m+1}$ will never result in a shortcut mutation that changes the probability function. In fact, for each $\mathtt{B}_i$, with $i \neq m+1$, there is only one production rule, and its duplication does not increase or decrease the probability of any string to be generated.

Because the shape of the derivation tree will always be the same, we know that for strings of length $n = 2^m$ the probability of selecting a node labeled $\mathtt{B}_{m+1}$ for mutation is $\frac{2^m}{2^{m+1}-1} > \frac{1}{2}$. This means that, in expectation, at least half of the times a mutation actually influencing the fitness of the grammar can happen. However, the only kind of mutation that can be accepted by the $(1+1)$-EA and that actually influences the quality of the grammar, is the insertion of another copy of the production rule $\mathtt{B}_{m+1} \rightarrow \mathtt{1}$. The average number of iterations to obtain this mutation (and, thus, a new grammar) is constant. In fact, at least one-half of the times a node labeled $\mathtt{B}_{m+1}$ is selected (which happens with a probability greater than $\frac{1}{2}$), a positive mutation event will occur with the insertion of another copy of the rule $\mathtt{B}_{m+1} \rightarrow \mathtt{1}$. The first of such mutations will increase the fitness of the grammar from $\frac{1}{2}n$ to $\frac{2}{3}n$ because the rule inserted can now be selected for the generation of the symbols in *all* positions of the strings (compare that with the grammar G_a, where each mutation was able to influence only one position). Because this argument can be iterated multiple times, we can state the following theorem:

Theorem 3. *For each $\ell \in \mathbb{N}^+$, starting from the grammar G_b defining the language L_n, the grammar generated by the $(1+1)$-EA employing the shortcut mutation of height 1 will have an expected quality after $\mathcal{O}(\ell)$ iterations greater than or equal to $\frac{\ell-1}{\ell}n$.* □

The situation with G_b is actually one of the best possible ones for the OneMax problem, because the degree of specialization is not linked with the size of the search space—a single mutation increases the probability of putting a $\mathtt{1}$ in each one of the n positions of the string.

3) Grammar G_c: Contrarily to G_b, the grammar G_c represents a worst case—that is, a grammar for which the $(1+1)$-EA is instead extremely slow in increasing the quality. As visible in Figure 1c, the G_c size is exponential with respect to n and is defined with a single non-terminal $\mathtt{C}$ (which is also the axiom) and by the following 2^n rules (all sharing the same LHS). It follows that any derivation tree generated with G_c will have one root node labeled with C with one child being a string in $\{\mathtt{0},\mathtt{1}\}^n$. This actually forces all mutations to happen at the root with a duplication of a rule $\mathtt{C} \rightarrow s$, which is accepted if and only if the fitness $|s|_1$ of s is greater than the current quality $q_f(G)$ of the grammar.

In this case, the *maximum* increase in fitness attainable in one iteration is given by always selecting the production rule $\mathtt{C} \rightarrow \mathtt{1}^n$ to be duplicated. Even if this happens at every mutation (recall, however, that this has a probability of $\frac{1}{2^n}$ to happen the first time), the resulting fitness after ℓ iterations would be the following:

$$\frac{n}{2} + \ell n \frac{1}{2^n} \tag{8}$$

This means that to reach a fitness of $\frac{2}{3}n$, an exponential number of successful mutation events must happen. We can then state the following result:

Theorem 4. *For each $\ell \in \mathbb{N}^+$, starting from the grammar G_c defining the language L_n, the grammar generated by the $(1+1)$-EA employing the shortcut mutation of height 1 will require* at least *an exponential number of iterations to move from quality $\frac{\ell}{\ell+1}n$ to $\frac{\ell+1}{\ell+2}n$.* □

In summary, we showed how the three grammars G_a, G_b, and G_c are differently susceptible to specialization in the OneMax problem when modified with the $(1+1)$-EA employing the shortcut mutation of height 1. In particular, G_b represents the best case, obtaining an expected quality of $\frac{\ell-1}{\ell}n$ after $\mathcal{O}(\ell)$ iterations; G_a represents an intermediate case, obtaining the same expected quality of $\frac{\ell-1}{\ell}n$ in at least ℓn and at most $\mathcal{O}(\ell n \log n)$ iterations; finally, G_c represents the worst case, obtaining the quality $\frac{\ell-1}{\ell}n$ not before $\mathcal{O}(\ell 2^{n-1})$ iterations.

G. Extension to the case of infinite binary strings

In this section, we show how the same kind of analysis done until now on a finite language can be extended to a simple infinite language.

To this end, we start by defining a new problem, OneMax$_\infty$, which has the same fitness function $f(s) = |s|_1$ of OneMax, but whose search space is the infinite language L_∞ of binary strings of unbounded length. Differently than OneMax, OneMax$_\infty$ has no global optimum.

We define the grammar G_∞ for L_∞ as:

$$\texttt{E} \rightarrow \texttt{0E} \mid \texttt{1E} \mid \lambda$$

where λ represents the empty string. It can be seen that G_∞ resembles G_a; moreover, the shape of the derivation trees generated with G_∞ is similar to the shape of those generated with G_a.

The probability of generating a string of length n with G_∞ is $\frac{1}{3}\left(\frac{2}{3}\right)^n$. The probability of generating a specific string of length n is $\frac{1}{2^n}\left(\frac{2}{3}\right)^n$. The quality of the grammar G_∞ is hence:

$$q_f(G_\infty) = \sum_{s \in L_\infty} p(s)|s|_1 = \frac{1}{3}\sum_{i=0}^{+\infty}\left(\frac{2}{3}\right)^i \frac{i}{2} = 1 \tag{9}$$

We can now derive a bound for the expected quality of the grammar generated by applying the $(1+1)$-EA of Algorithm 1 to G_∞. We start by observing that in the derivation tree of any string of length n, there are $n+1$ nodes labeled with E, one of which has a single child labeled with λ. On average, hence, $\frac{n}{2}$ nodes in the derivation tree will have 1 as the leftmost child and $\frac{n}{2}$ will have 0 as the leftmost child. Because a successful mutation occurs when a node with the one as a child is chosen by the shortcut mutation, the probability of a successful mutation is the following:

$$\sum_{i=0}^{+\infty}\frac{i}{2i+1}\left(\frac{2}{3}\right)^i\frac{1}{3} \geq \frac{1}{3}\sum_{i=0}^{+\infty}\frac{i}{2i+2}\left(\frac{2}{3}\right)^i = \frac{1}{2} - \frac{\log_e 3}{4} \tag{10}$$

The first successful mutation produces a grammar G in which the average fitness for a string of length n is $\frac{2}{3}n$ and, in general, after h successful mutations, the average fitness among the strings of length n will be $\frac{h+1}{h+2}n$. The average quality of the resulting grammar will then be:

$$\begin{aligned} q_f(G) &= \sum_{i=0}^{+\infty}\frac{1}{h+3}\left(\frac{h+2}{h+3}\right)^i\frac{h+2}{h+3}i \\ &= \frac{h+2}{(h+3)^2}\sum_{i=0}^{+\infty}\left(\frac{h+2}{h+3}\right)^i i \end{aligned}$$

By recalling that $\sum_{k=1}^{+\infty} kx^k = x(x-1)^{-2}$ for $|x| < 1$, which holds in this case, we can continue as follows:

$$\begin{aligned} &= \frac{h+2}{(h+3)^2}\frac{h+2}{h+3}\left(\frac{h+2}{h+3}-1\right)^{-2} \\ &= \frac{(h+2)^2}{h+3} \end{aligned} \tag{11}$$

This shows that the quality of the grammar increases almost linearly with the number of successful mutations (i.e., the order is $h+2$ multiplied by a factor that goes to one). Because the probability of obtaining a successful mutation is at least $\frac{1}{2} - \frac{\log_e 3}{4}$, a successful mutation happens, on average, after a constant number of iterations. We can, therefore, state the following theorem:

Theorem 5. *Starting from the grammar G_∞ defining the language L_∞, the grammar generated by the $(1+1)$-EA employing the shortcut mutation of height 1 has its expected fitness bounded above by $\mathcal{O}(t)$ after t iterations.*

IV. Experimental analysis

In this section, we present the experimental analysis we conducted to validate the findings of the theoretical analysis of Section III and to further extend them. More precisely, we aimed to answer the following research questions:

1) Is the ranking concerning the proneness to specialization in OneMax among G_a, G_b, G_c the same in theory and in experiments?
2) Is that ranking qualitatively confirmed when other grammars (obtained by "mixing" G_a, G_b, G_c) are considered for the same problem?
3) What happens when one uses an unbounded shortcut mutation, instead of a mutation with height 1, with the same grammars and problem?
4) What happens in the case of different problems?

For this purpose, we considered the same scenario of Section III-F—that is, the application of the $(1+1)$-EA of Algorithm 1 employing the shortcut mutation for specializing a grammar in a problem. We performed several experiments by considering various shortcut mutations (height 1 or unbound), problems, and grammars.

An important difference from the theoretical study is that the quality $q_f(G)$ of a grammar G for a problem defined by f is *not* computed analytically; instead, here, we use an empirical estimate obtained by generating 1000 strings from a uniform distribution, and computing the average fitness of these strings. Notice that in the experiments there are, in some cases, fewer than 1000 possible strings generated by the grammar. Still, we need to estimate the probability distribution with which the strings are generated: the accuracy of the estimate increases with the number of samples. Using an estimate of the quality instead of the actual value might result in accepting some mutations that should be rejected or rejecting mutations that actually have a positive effect on the quality.

A. Problems

We here call problem a pair composed of a language L and a fitness function $f : L \rightarrow \mathbb{R}$. We experimented with six problems obtained by considering two languages and three fitness functions.

Concerning the languages, we considered the language of the binary strings of length n, i.e., $L = \{\texttt{0}, \texttt{1}\}^n$, and language of the ternary strings of length n, i.e., $L = \{\texttt{0}, \texttt{1}, \texttt{2}\}^n$. In the previous sections, we defined three grammars for the binary case; the corresponding grammars for the ternary case can be easily obtained. In the following, we will use G_a, G_b, and G_c

$$
\begin{aligned}
&\mathtt{A}_1 \to \mathtt{0A}_2 \mid \mathtt{1A}_2\\
&\mathtt{A}_2 \to \mathtt{0A}_3 \mid \mathtt{1A}_3\\
&\dots\\
&\mathtt{A}_k \to \mathtt{0B}_1 \mid \mathtt{1B}_1\\
&\mathtt{B}_1 \to \mathtt{B}_2\mathtt{B}_2\\
&\dots\\
&\mathtt{B}_{m+1} \to \mathtt{0} \mid \mathtt{1}
\end{aligned}
$$

(a) $G^k_{a,b}$ grammars.

$$
\begin{aligned}
&\mathtt{A}_1 \to \mathtt{0A}_2 \mid \mathtt{1A}_2\\
&\mathtt{A}_2 \to \mathtt{0A}_3 \mid \mathtt{1A}_3\\
&\dots\\
&\mathtt{A}_k \to \mathtt{0C} \mid \mathtt{1C}\\
&\mathtt{C} \to \mathtt{0}\dots\mathtt{0} \mid \mathtt{0}\dots\mathtt{1} \mid\\
&\qquad \dots \mid \underbrace{\mathtt{1}\dots\mathtt{1}}_{\in\{\mathtt{0},\mathtt{1}\}^{n-k}}
\end{aligned}
$$

(b) $G^k_{a,c}$ grammars.

Fig. 3: Intermediate grammars $G^k_{a,b}$ and $G^k_{a,c}$ for the binary strings of length n.

for both the binary and the ternary version when no confusion is possible.

Concerning the fitness functions, we considered the following three cases:

- ONEMAX. In both the binary and the ternary case, the fitness of a string is computed as the number of ones it contains. This problem has a single global optima in 1^n.
- DIFFERENCE. Fitness f of string s over alphabet T is computed as
$$f(s) = \max_{\alpha_1,\alpha_2 \in T}\{|s|_{\alpha_1} - |s|_{\alpha_2}\}$$
where $|s|_\alpha$ for $\alpha \in T$ denotes the number of α in string s. This function has $|T|$ global optima, corresponding to all of the strings of form α^n for $\alpha \in T$.
- POSITIONAL. Similarly to ONEMAX, the fitness of string s is computed as the Hamming distance between s and a target string, except that in this case the target string instead of being 1^n, can be different. We employed `01010101` for the binary case and `01201201` for the ternary case.

Note that the case we analyzed theoretically is the one corresponding to the binary ONEMAX.

B. Intermediate grammars

In addition to grammars G_a, G_b, and G_c, we consider two classes of intermediate grammars; every intermediate grammar still defines the language L_n of the binary strings of length n defined by grammars G_a, G_b, G_c. In the first class, a grammar $G^k_{a,b}$ is designed to be a mix of the grammars G_a and G_b: when generating a string, $0 \le k \le n$ symbols in the string will be generated with a portion of the grammar similar to G_a, and $n-k$ symbols with a portion similar to G_b. Similarly, in the second class, a grammar $G^k_{a,c}$ is designed to be a mix of the grammars G_a and G_c: when generating a string, $0 \le k \le n$ symbols in the string will be generated with a portion of the grammar similar to G_a, and $n-k$ symbols with a portion similar to G_c. Extreme cases with respect to parameter k correspond to the base grammars—that is, $G^0_{a,b} = G_b$, $G^n_{a,b} = G_a$, $G^0_{a,c} = G_c$, and $G^n_{a,c} = G_a$.

Figure 3 shows the two classes of grammars for the binary case. For the ternary case, we do not show the full grammars for brevity: a few modifications are, however, needed with respect to the binary case. In the part related to G_a, the following production rules are added:

$$
\begin{aligned}
&\mathtt{A}_k \to \mathtt{2A}_{k+1} \qquad &&\text{for } 1 \le k < n\\
&\mathtt{A}_n \to \mathtt{2}
\end{aligned}
$$

In the part related to G_b, only the production rule $\mathtt{B}_m \to \mathtt{2}$ is added. In the part related to G_c, all of the production rules $\mathtt{C} \to s$, for all $s \in \{\mathtt{0},\mathtt{1},\mathtt{2}\}^n$ with $|s|_2 > 0$—that is, all words not already in $\{0,1\}^n$ are added.

C. Discussion of the results

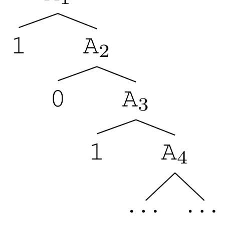
We performed 100 runs of the $(1+1)$-EA of Algorithm 1 with $l = 100$ iterations for each mutation, language, fitness, and starting grammar (i.e., 100 runs for each of the $2\times2\times3\times15 = 180$ combinations). Figure 4 presents the results for the binary grammars, and Figure 5 presents the results for the ternary grammars in terms of the average grammar quality across the 100 runs during the evolution.

As a first observation, we can notice that the behavior for the binary and ternary case is very similar, particularly with respect to the ranking of the various grammars. This behavior seems to indicate that small variations in the alphabet size do not influence the general behaviors of the grammars.

We now discuss the results for the three fitness functions and two mutations.

1) ONEMAX*:* As it is possible to observe from the plot, using height 1 shortcut mutation produces a ranking among the three "main" grammars G_a, G_b, and G_c that respects the theoretical predictions: an increase in the quality of G_b is more pronounced compared with the one in G_a, whereas G_c remains almost stationary at the initial value, with no significant improvements. The intermediate grammars $G^k_{a,b}$ also have a ranking given by the value of k, with grammars that are more "similar" to G_b specializing faster than the ones more "similar" to G_a. A similar ranking was also obtained for grammars $G^k_{a,c}$.

When moving to the unbounded shortcut mutation, we observe that the ranking among the various grammars has not changed, but there is a significant difference in the behavior. First of all, there is a wide gap between G_c and all $G^k_{a,c}$ grammars, which shows that increasing the possible height of the mutation has a beneficial effect even for grammars that are minimally different from G_c. The other important observation is that the increase in quality seems to slow down earlier for both G_a, and G_b (and the intermediate grammars between them). A possible explanation for this type of behavior is the following. Consider, for example, the grammar G_a and the following derivation tree for G_a (pruned for brevity):



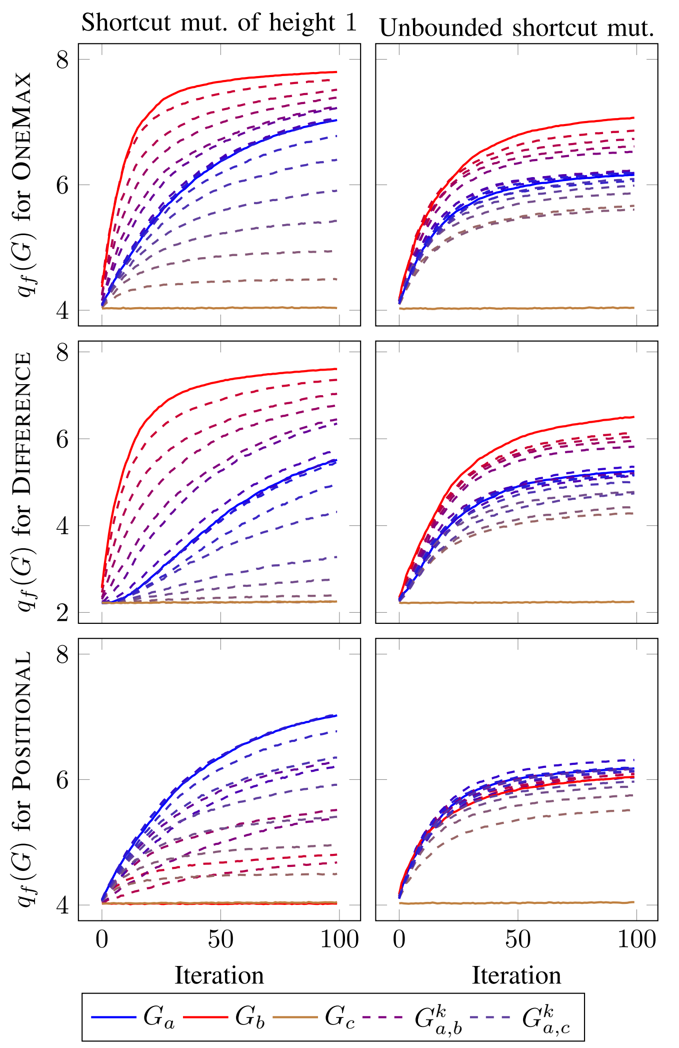


Fig. 4: Fitness of the $(1+1)$-EA, i.e., quality $q_f(G)$ of the grammar, during the evolution, averaged across the 100 runs in the *binary* case for different initial grammars (line of plot), fitness functions (row of plots), and shortcut mutations (column of plots).

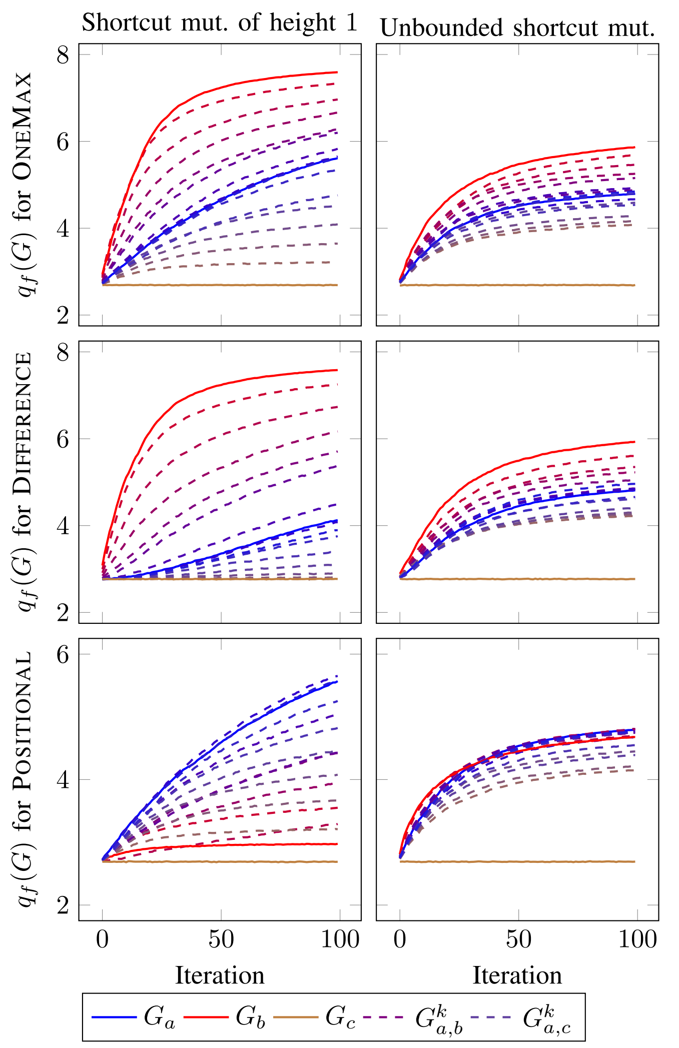


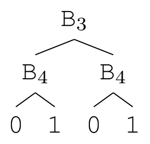
Fig. 5: Fitness of the $(1+1)$-EA, i.e., quality $q_f(G)$ of the grammar, during the evolution, averaged across the 100 runs in the *ternary* case for different initial grammars (line of plot), fitness functions (row of plots), and shortcut mutations (column of plots).

If a connected subset τ of the tree is selected such that the corresponding rule is $\texttt{A}_1 \to \texttt{101A}_4$, then the mutation will be accepted, as the average number of ones in a string increases when that rule is used. However, the rule also increases the probability of having a zero in the second position of the generated string. In fact, with probability $\frac{1}{3}$ at the first step of the generation of a string the rule $\texttt{A}_1 \to \texttt{101A}_4$ is selected (and with probability $\frac{2}{3}$ the generation proceeds without the newly added derivation rule). This means that the probability of obtaining 0 in the second position of the string is now $\frac{1}{3} + \frac{2}{3}\frac{1}{2} = \frac{2}{3}$, which is an increase with respect to the original value $\frac{1}{2}$. Although the quality will continue to increase for a while, the compound effect of mutations such as this one can probably slow down the specialization process.

2) DIFFERENTIAL*:* The results for the differential fitness function are quite similar to the ones observed for the ONEMAX function, even if the problem has multiple global optima. The main difference can be observed for the grammar G_a and the intermediate grammars of the form $G^k_{a,c}$ in the ternary case for the mutation of height 1. A possible explanation for this kind of behavior is that, at least at the beginning of the search process, there are multiple ways in which to increase the average fitness of the grammar by producing slight imbalances among the number of $\texttt{0}$s, $\texttt{1}$s, and $\texttt{2}$s. Only when one of the symbols in the alphabet has reached "critical mass" all successive mutations are toward a specific optimum.

3) POSITIONAL*:* The results for this particular problem show a very large difference between the mutation of height 1 and the unbounded case. In the case of height 1 mutation there is no improvement in the average fitness of the grammar for G_b. This can be explained in the following way: the grammar G_b has a single non-terminal used to directly generate all terminals in all positions: B_m is either rewritten in $\texttt{0}$, if the production rule $\texttt{G}_m \to \texttt{0}$ is applied, or in $\texttt{1}$, if the production rule $\texttt{G}_m \to \texttt{1}$ is applied instead. If rule $\texttt{G}_m \to \texttt{0}$ has been selected to be duplicated, then the probability of generating a $\texttt{0}$ in *all* positions of the string increases. However, because the target contains the same number of `0`s and `1`s, there is no increase in fitness. A similar reasoning holds for the duplication of the rule $\texttt{G}_m \to \texttt{1}$. In this sense, grammar G_b not only remains stable in its quality in the measured 100 iterations but also it *cannot* specialize at all in this problem. This is actually worse than grammar G_c, which can still specialize, even if the specialization is exponentially slow.

The same behavior does not appear in the unbounded height case. This can be explained by the existence of a mutation that *can* increase the quality, as in the following case. Let τ be the following connected subset of a derivation tree:



If it is selected for mutation, τ would generate the production rule $\texttt{B}_3 \to \texttt{0101}$, and because it represents "half" of the optimum, when inserted into G_b it will increase the quality of the grammar. In some sense, the mutation of height larger than 1 can "break the symmetry" of the production rules $\texttt{G}_m \to \texttt{0}$ and $\texttt{G}_m \to \texttt{1}$, which made it impossible for the mutation of height 1 to increase the quality. Interestingly, this beneficial instance of shortcut mutation actually corresponds to finding a useful building block for the POSITIONAL problem.

V. CONCLUDING REMARKS

With reference to a context of evolutionary optimization, we considered the possibility of specializing a grammar in a problem. We defined a way to quantify the quality of a grammar with respect to a problem in a way that is meaningful for an EA, and proposed a shortcut mutation operator for modifying a grammar without altering the corresponding language. We showed how the proposed mutation operator can be employed by a $(1+1)$-EA to specialize a grammar in a problem, i.e., to increase the average fitness of the candidate solutions to the problem generated by the grammar.

We analyzed theoretically the particular case in which the problem is ONEMAX and the shortcut mutation results in varying the probabilities of selecting the production rules. In this context, we considered three grammars that define the same finite language of the binary strings of length n (i.e., the one of the ONEMAX problem), and we showed that these grammars behave differently when evolved with the $(1+1)$-EA. In particular, we found that the expected number of iterations for reaching the quality $\frac{\ell}{\ell+1}n$ (for $\ell \geq 1$) can be: constant with respect to n and linear with respect to ℓ (Theorem 3), bounded between polynomials in both n and ℓ (Theorems 1 and 2), or exponential with respect to n (Theorem 4). We also showed that a similar analysis can be extended to infinite languages.

Then, we experimentally validated the theoretical findings (from a qualitative perspective) and extended them in several ways, by considering more grammars, more problems, and an unbounded version of the shortcut mutation that may result in finding building blocks. We considered problems where solutions are either binary or ternary strings and in which the fitness may depend on the position of the symbols. The experimental results confirmed the theoretical findings and showed that the degree to which a grammar can be specialized depends on the problem and is related to solution structures favored by the fitness function and more or less expressible by the grammar upon specialization. We speculate that similar results might be obtained, using the same theoretical framework, by evolving the grammar with a different EA than the $(1+1)$-EA here considered: we leave this to future work.

We believe that our results can foster further research about grammar specialization and design, of both theoretical and practical nature.

Concerning the former, it would be interesting to devise a more comprehensive definition of the quality of a grammar: our definition corresponds, in some sense, to the expected quality of the initial population that uses that grammar to generate the candidate solutions; one could instead attempt to capture the overall expected dynamics of an EA (e.g., a $(1+1)$-EA) operating with the grammar, rather than considering only the initial population. Moreover, a finer theoretical characterization of which problem features (e.g., multi-modality, deceptiveness) impact the possibility of specializing a grammar would be beneficial: the experiments described in this study already suggest that problem features are relevant. From a broader point of view, being able to characterize and measure the complexity of the fitness landscape induced by a grammar might be useful for driving the optimization of the grammar itself.

On the practical side, it would be interesting to investigate whether the geometrical formulation of semantic mutation and crossover (as defined in [51]) could be extended to handle *entire* grammars, possibly building on the recently defined extension of semantic operators for GE [52].

ACKNOWLEDGMENT

This work was supported by national funds through FCT (Fundação para a Ciência e a Tecnologia) by the project GADgET (DSAIPA/DS/0022/2018) and AICE (DSAIPA/DS/0113/2019). This work was also partially supported by projects UID/MULTI/00308/2019 and by the European Regional Development Fund through the COMPETE 2020 Programme, FCT - Portuguese Foundation for Science and Technology and Regional Operational Program of the Center Region (CENTRO2020) within project MAnAGER (POCI-01-0145-FEDER-028040). Mauro Castelli acknowledges the financial support from the Slovenian Research Agency (research core funding No. P5-0410).

REFERENCES

[1] S. Ginsburg, *The Mathematical Theory of Context Free Languages.* McGraw-Hill Book Company, 1966.
[2] P. A. Whigham, "Grammatically-based genetic programming," in *Proceedings of the workshop on genetic programming: from theory to real-world applications*, vol. 16, no. 3, 1995, pp. 33–41.
[3] C. Ryan, J. Collins, and M. O'Neill, *Grammatical evolution: Evolving programs for an arbitrary language*. Springer, 1998, pp. 83–96.
[4] 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.
[5] A. Bartoli, M. Castelli, and E. Medvet, "Weighted Hierarchical Grammatical Evolution," *IEEE Transactions on Cybernetics*, pp. 1–13, 2018.

[6] P. A. Whigham, G. Dick, J. Maclaurin, and C. A. Owen, "Examining the best of both worlds of grammatical evolution," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, 2015, pp. 1111–1118.

[7] L. Spector, "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*, vol. 18, no. 3, pp. 351–352, 2017.

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

[9] K. De Jong, "Parameter setting in EAs: a 30 year perspective," *Parameter setting in evolutionary algorithms*, pp. 1–18, 2007.

[10] B. Manderick, M. K. de Weger, and P. Spiessens, "The genetic algorithm and the structure of fitness landscape," in *4th International Conference on Genetic Algorithms (ICGA)*, 1991, pp. 143–150.

[11] E. Pitzer and M. Affenzeller, "A comprehensive survey on fitness landscape analysis," in *Recent advances in intelligent engineering systems*. Springer, 2012, pp. 161–191.

[12] M. Nicolau and A. Agapitos, "Understanding grammatical evolution: Grammar design," *Handbook of Grammatical Evolution*, pp. 23–53, 2018.

[13] T. Saber, D. Fagan, D. Lynch, S. Kucera, H. Claussen, and M. O'Neill, "Multi-level grammar genetic programming for scheduling in heterogeneous networks," in *Genetic Programming*, M. Castelli, L. Sekanina, M. Zhang, S. Cagnoni, and P. García-Sánchez, Eds. Springer, 2018, pp. 118–134.

[14] R. Loughran and M. O'Neill, "Serendipity in Melodic Self-organising Fitness," in *AISB 2018 Symposium: Cybernetic Serendipity Reimagined.*

[15] M. Nicolau, "Understanding grammatical evolution: initialisation," *Genetic Programming and Evolvable Machines*, vol. 18, no. 4, pp. 467–507, 2017.

[16] 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, 2011.

[17] P. J. Angeline and J. B. Pollack, "Coevolving high-level representations," in *Santa Fe Institute Studies in the Sciences of Complexity -Proceedings Volume-*, vol. 17. Addison-Wesley Publishing Co, 1994, pp. 55–55.

[18] J. P. Rosca and D. H. Ballard, *Genetic programming with adaptive representations*. University of Rochester, Department of Computer Science, 1994.

[19] P. J. Angeline, "Two self-adaptive crossover operators for genetic programming," *Advances in Genetic Programming*, vol. 2, pp. 89–109, 1996.

[20] H. Iba and H. de Garis, "Extending genetic programming with recombinative guidance," *Advances in genetic programming*, vol. 2, pp. 69–88, 1996.

[21] 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.

[22] H. Mühlenbein and G. Paass, "From recombination of genes to the estimation of distributions I. binary parameters," in *International conference on parallel problem solving from nature*. Springer, 1996, pp. 178–187.

[23] J. S. De Bonet, C. L. Isbell, Jr., and P. Viola, "Mimic: Finding optima by estimating probability densities," in *Proceedings of the 9th International Conference on Neural Information Processing Systems (NIPS'96)*. MIT Press, 1996, pp. 424–430.

[24] R. Etxeberria and P. Larrañaga, "Global optimization using bayesian networks," in *Proc. 2nd Symposium on Artificial Intelligence (CIMAF-99)*, 1999.

[25] R. S. Michalski, "Learnable evolution model: Evolutionary processes guided by machine learning," *Machine learning*, vol. 38, no. 1-2, pp. 9–40, 2000.

[26] R. Salustowicz and J. Schmidhuber, "Probabilistic incremental program evolution," *Evolutionary Computation*, vol. 5, no. 2, pp. 123–141, 1997.

[27] S. Baluja, "Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning," Carnegie-Mellon Univ Pittsburgh Pa Dept Of Computer Science, Tech. Rep., 1994.

[28] K. Yanai and H. Iba, "Estimation of distribution programming based on bayesian network," in *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.*, vol. 3. IEEE, 2003, pp. 1618–1625.

[29] P. A. Whigham, "Inductive bias and genetic programming," in *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, 1995, pp. 461–466.

[30] P. A. Whigham, "Search bias, language bias and genetic programming," in *Proceedings of the 1st annual conference on genetic programming*. MIT Press, 1996, pp. 230–237.

[31] A. Ratle and M. Sebag, "Avoiding the bloat with stochastic grammar-based genetic programming," in *International Conference on Artificial Evolution (Evolution Artificielle)*. Springer, 2001, pp. 255–266.

[32] H. A. Abbass, X. Hoai, and R. I. Mckay, "Anttag: A new method to compose computer programs using colonies of ants," in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02*, vol. 2. IEEE, 2002, pp. 1654–1659.

[33] M. Dorigo and T. Stützle, "The ant colony optimization metaheuristic: Algorithms, applications, and advances," in *Handbook of metaheuristics*. Springer, 2003, pp. 250–285.

[34] R. Shan, R. McKay, H. Abbass, and D. Essam, "Program evolution with explicit learning," in *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.*, vol. 3. IEEE, 2003, pp. 1639–1646.

[35] P. A. N. Bosman and E. D. de Jong, "Grammar transformations in an EDA for genetic programming," Department of Information and Computing Sciences, Utrecht University, The Netherlands, Tech. Rep., 2004.

[36] Y. Shan, R. I. McKay, R. Baxter, H. Abbass, D. Essam, and H. Nguyen, "Grammar model-based program evolution," in *Proceedings of the 2004 Congress on Evolutionary Computation*, vol. 1. IEEE, 2004, pp. 478–485.

[37] K. Kim, Y. Shan, X. H. Nguyen, and R. I. McKay, "Probabilistic model building in genetic programming: a critical review," *Genetic Programming and Evolvable Machines*, vol. 15, no. 2, pp. 115–167, Jun 2014.

[38] M. Toussaint, "Self-adaptive exploration in evolutionary search," *arXiv preprint physics/0102009*, 2001.

[39] O. A. Palacios, C. R. Stephens, and H. Waelbroeck, "Emergence of algorithmic language in genetic systems," *Biosystems*, vol. 47, no. 3, pp. 129–147, 1998.

[40] 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.

[41] K. L. Nickerson, Y. Chen, F. Wang, and T. Hu, "Measuring evolvability and accessibility using the hyperlink-induced topic search algorithm," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2018, pp. 1175–1182.

[42] M. O'Neill and C. Ryan, "Grammatical evolution by grammatical evolution: The evolution of grammar and genetic code," in *Genetic Programming*, M. Keijzer, U.-M. O'Reilly, S. Lucas, E. Costa, and T. Soule, Eds. Springer, 2004, pp. 138–149.

[43] M. O'Neill and A. Brabazon, "mGGA: The meta-grammar genetic algorithm," in *Genetic Programming*, M. Keijzer, A. Tettamanzi, P. Collet, J. van Hemert, and M. Tomassini, Eds. Springer, 2005, pp. 311–320.

[44] E. Hemberg, M. O'Neill, and A. Brabazon, "Grammatical bias and building blocks in meta-grammar grammatical evolution," in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. IEEE, 2008, pp. 3775–3782.

[45] M. Nicolau, "Automatic grammar complexity reduction in grammatical evolution," in *The 3rd Grammatical Evolution Workshop: A workshop of the 2004 Genetic and Evolutionary Computation Conference (GECCO-2004), Seattle, Washington, USA, 26-30 June 2004*, 2004.

[46] E. Hemberg, N. McPhee, M. O'Neill, and A. Brabazon, "Pre-, in-and postfix grammars for symbolic regression in grammatical evolution," in *IEEE Workshop and Summer School on Evolutionary Computing*, 2008, pp. 18–22.

[47] P. S. Oliveto and X. Yao, "Runtime analysis of evolutionary algorithms for discrete optimization," in *Theory of Randomized Search Heuristics: Foundations and Recent Developments*. World Scientific, 2011, pp. 21–52.

[48] S. Droste, T. Jansen, and I. Wegener, "On the analysis of the (1+1) evolutionary algorithm," *Theoretical Computer Science*, vol. 276, no. 1-2, pp. 51–81, 2002.

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

[50] P.-K. Wong, M.-L. Wong, and K.-S. Leung, "Hierarchical Knowledge in Self-Improving Grammar-Based Genetic Programming," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2016, pp. 270–280.

[51] A. Moraglio, K. Krawiec, and C. G. Johnson, "Geometric semantic genetic programming," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2012, pp. 21–31.

[52] A. Moraglio, J. McDermott, and M. O'Neill, *Geometric semantic grammatical evolution*. Springer, 2018, pp. 163–188.