Grammatically uniform population initialization for grammar-guided genetic programming

Pablo Ramos Criado · Dolores Barrios Rolanía · Daniel Manrique · Emilio Serrano

Abstract The initial population distribution is an essential issue in evolutionary computation performance. Population initialization methods for grammar guided genetic programming have some difficulties generating a representative sample of the search space, which negatively affects the overall evolutionary process. This paper presents a grammatically uniform population initialization method to address this issue by improving the initial population uniformity: the equiprobability of obtaining any individual of the search space defined by the context-free grammar. The proposed initialization method assigns and updates probabilities dynamically to the production rules of the grammar to pursue uniformity, and includes a code bloat control mechanism. We have conducted empirical experiments to compare the proposed algorithm with a standard initialization approach very often used in grammar-guided genetic programming. The results report that the proposed initialization method approximates very well a uniform distribution of the individuals in the search space. Moreover, the overall evolutionary process that takes place after the population initialization performs better in terms of convergence speed and quality of the final solutions achieved when the proposed method generates the initial population than when the usual approach does. The results also show that these performance differences are more significant when the experiments involve large search spaces.

Keywords Grammar-guided genetic programming · Initialization · Genotypic uniformity · Stochastic context-free grammar

P. Ramos Criado
Aturing Research, Salamanca, Spain
E-mail: pablo.ramos@aturing.com

D. Barrios Rolanía
Departamento de Matemáticas del Área Industrial
Universidad Politécnica de Madrid, Spain
E-mail: dolores.barrios.rolania@upm.es

D. Manrique
Departamento de Inteligencia Artificial
Universidad Politécnica de Madrid, Spain
ORCID ID: 0000-0002-0792-4156 E-mail:
daniel.manrique@upm.es

E. Serrano
Departamento de Inteligencia Artificial
Universidad Politécnica de Madrid, Spain
ORCID ID: 0000-0001-7587-0703 E-mail:
emilioserra@fi.upm.es

1 Introduction

Evolutionary computation (Kari and Rozenberg, 2008) is a subfield of Natural Computing that borrows ideas from natural evolution (Darwin, 1959) to perform search and optimization processes by evolving populations of individuals (subsets of the search space) that represent candidate solutions to a specific problem (solution space). Genetic programming (Koza, 1992) and grammar-guided genetic programming (GGGP) (Whigham, 1995) are evolutionary algorithms that belong to evolutionary computation. GGGP is an extension of genetic programming designed to optimize programs belonging to a search space defined by a context-free grammar (CFG) (Hopcroft et al., 2006; Sipser, 2013; Krithivasan, 2009; Moll et al., 2012). The CFG establishes the set of syntactical restrictions that all individuals, namely, derivations or parse trees, have to meet. This way, the closure problem is overcome, which assures that individuals are feasible (McKay et al., 2010).

Similar to other evolutionary algorithms, GGGP starts generating the initial population following a particular distribution (García Arnau et al., 2007). Once

the population is initialized, the evolutionary process begins. It comprises three primary operations executed in a loop: selection, variation, and replacement. The selection operation chooses some promising individuals of the population, called parents. The variation operation generates a set of new individuals, called offspring, based on the genotypic characteristics of the parents. Then, the replacement operator inserts these new individuals into the population and discards other less promising individuals for the new generation. A fitness function is applied each time an individual is generated, after the initialization process or variation operation, to measure how well the candidate solution encoded by the individual fits the problem. The evolutionary loop stops when the population meets a convergence criterium. The best or some of the individuals of the final population is expected to solve the problem.

In order to improve the performance of optimization algorithms, mainly to solve large-scale problems, some research works focus on finding an adequate trade-off between expand and local search (Deng et al., 2019). If the algorithm focuses on speeding up convergence, then it likely converges to a local optimum. On the contrary, if the algorithm explores too much the search space, the converge speed slows down, preventing the optimization process from reaching a quality solution. Intending to provide such balance, collaborative or hybrid strategies combine different meta-heuristics like genetic algorithms and ant colony optimization (Deng et al., 2016). Particle swarm optimization is another approach that employs the alpha-stable distribution theory to replace the uniform distribution to escape from local minima, and dynamic fractional calculus with memory characteristic to improve the convergence speed (Deng et al., 2017).

1.1 Initial population and performance

Besides specifically designed optimization approaches to balance the exploration and exploitation capabil-ities of the optimization process, the performance of evolutionary algorithms is also affected by how representative of the search space the initial population is (Harper, 2010), independently of the variation operators involved in the evolutionary process. Diverse ini-tial populations, where the individuals are well-spread along with the search space, improve the evolutionary optimization process. On the contrary, a reduced diversity at the beginning of the evolution process may produce premature convergence to local optima (Burke et al., 2004; García Arnau et al., 2007). There are several proposals to measure population diversity in evolutionary algorithms (Nicolau, 2017). These proposals can be classified into structural and phenotypic diversity measurements. Structural diversity is mainly related to genotype diversity, while phenotypic diversity is more related to behavior or fitness (Hien and Hoai, 2006). Since two different genotypes may represent the same phenotype in GGGP, the structural diversity may be distorted. In particular, the use of ambiguous CFG, where two different derivation trees can encode the same solution, may cause this distortion (Nicolau and Fenton, 2016). In this case, phenotypic diversity measurement seems to fit better than the genotype-based diversity. However, it is problem-dependent and not accurate.

The initial population is usually generated at random (Hassanat et al., 2018). Random branch is a significant approach that includes a bloat control mechanism, although it cannot reach all derivation trees in the search space (Chellapilla, 1997; Koza et al., 2006; McKay et al., 2010). However, random not always means uniform, and the initialization process should be driven to achieve representative population samples. Several approaches claim uniform population initialization as the exact uniform tree generation algorithm, which takes a requested tree size and creates a tree chosen uniformly from the full set of all possible trees of that size. These approaches involve methods that bias the population initialization to some specific individual sizes that improve uniformity (Vanneschi et al., 2014; Murphy et al., 2012; Tanev, 2004). The depth and breadth of derivation trees are usually the targets of uniformity search (Poli et al., 2008).

GGGP approaches deal with inherent CFG additional constraints that make obtaining a specific distribution more difficult. Therefore, the CFG must be considered to design a process that generates uniform populations (Fagan et al., 2016). Every CFG possesses different production rules and symbols that bias the population initialization process in different ways (Murphy et al., 2012; Thorhauer, 2016; Schweim et al., 2018). Some GGGP population initialization approaches analyze the CFG characteristics to describe how derivation trees are produced and guide the population initialization process (Ramos Criado, 2017). However, examining CFG does not achieve quite a uniformity in initial populations. GGGP research has also focused on including size bounds to reduce code bloat (McKay et al., 2010; Crane and McPhee, 2006). However, size bounds can be inappropriate for CFG since they are not strictly related to the search space or how derivation trees grow. Different depth bounds may not vary the set of derivation trees within the search space.

1.2 Contributions

A grammatically uniform population initialization (GUPI) for grammar-guided genetic programming is presented to improve the initial population uniformity and to provide more proper bounds that reduce the code bloat. This new approach refers to uniformity as the genotypic, structural, or grammatical type: the equiprobability of obtaining any individual of the search space defined by the CFG. Since the encoding scheme establishes a relationship between the search and solution spaces, grammatically uniform initial populations tend to increase phenotypic diversity at the beginning of the evolutionary process. The proposed population initialization is driven by a dynamic stochastic CFG to achieve uniformity. The probabilities are dynamically assigned to shape population initialization according to the CFG characteristics and each derivation tree construction process. Additionally, dynamic recursion bounds are included to control code bloat.

Empirical experiments have been conducted to compare the initial population distribution and the performance of the overall GGGP evolutionary process when starting from initial populations generated by GUPI and a standard frequently used initialization algorithm with code bloat control that reports good results for GGGP (García Arnau et al., 2007). Two very often vari-ation operators in GGGP, but with very different be-havior, have been considered: generational replacement estimation of distribution algorithm (EDA) (Kim et al., 2014) and Whigham's crossover (WX) (Whigham, 1995). While generational replacement EDA primarily performs a local search, WX tends to explore the search space widely. The results report that the proposed initializa-tion approach generates individuals following a uniform distribution in the search space. Both initial population uniformity and code bloat control drive GUPI to popu-lation distributions that improve the overall evolution-ary process performance, compared to general initial populations, even if recursive CFG are employed.

The rest of the paper is structured as follows: Section 2 defines the cardinality of a production rule or symbol, the probability of a derivation, and the average probability of a terminal string derivation. These new concepts allow understanding how and why initial populations are usually biased to the smallest derivation trees in GGGP, especially with recursive CFG. This section also discusses, based on these concepts, that using stochastic CFG is not either a feasible solution. Section 3 describes the main contribution of the paper, the grammatically uniform population initialization algorithm along with an example. The experimental results section firstly compares the distribution of the individuals generated by GUPI and the standard initialization approach for three different problems. Then, performance results have been gathered over six experiments for each problem to compare the overall GGGP evolutionary process starting from initial populations generated by GUPI and the standard approach. Finally, Section 5 provides some concluding remarks, contributions, and future lines of research.

2 Problem description

Let $G = (V, \Sigma, R, S)$ be a CFG, where Σ is the set of terminal symbols, V is the set of nonterminal symbols, S is the axiom or start symbol of the grammar and R is the set of production rules of the form $A ::= \gamma$, such that $A \in V$ and $\gamma \in (V \cup \Sigma)^*$. The asterisk represents the Kleene closure operation. $R_A \subset R$ denotes the set of all production rules with the same left-hand side nonterminal symbol A, $R_A = \{r : A ::= \gamma, \gamma \in (\Sigma \cup V)^*\}$.

The cardinality of a production rule or symbol is the number of different terminals string derivations $A \overset{+}{\Rightarrow} s$, $s \in \Sigma^*$, that can be produced, starting from that production rule or symbol. $\overset{+}{\Rightarrow}$ denotes the transitive closure of $\Rightarrow$, where one or more production rules are applied. $|x|$ denotes the cardinality of a production rule or symbol x. Calculating $|x|$ involves the following notation: let $V_0, V_1, ..., V_n \subset V$,

1. $V_0 = \{A \in V : \forall r \in R_A, \quad r : A ::= s, s \in \Sigma^*\}$.
2. For $i = 1, ..., n$:
   $V_i = \{A \in V : \forall r \in R_A, \quad r : A ::= \gamma, \gamma \in (\Sigma \cup V_{i-1})^*\}$.

Then, $|x|$ is calculated as follows:

1. $|a| = 1, \forall a \in \Sigma$.
2. For $i = 0, 1, ..., n$, if $r \in R_A$, $r : A ::= \gamma$, $A \in V_i$ and $\gamma = B_0 B_1 ... B_m$, then:

   $$|r| = \prod_{k=0}^{m} |B_k|.$$

3. For $i = 0, 1, ..., n$, if $A \in V_i$:

   $$|A| = \sum_{r \in R_A} |r|.$$

If the CFG is unambiguous, $|x|$ also equals the number of different terminal strings yielded from x. The cardinality of non-recursive production rules is finite, while the cardinality of recursive production rules is infinite. This research work considers the general case of approaches with code bloat control mechanism where the derivation steps are limited. Under these conditions, the production rules cardinality is always finite, even for

recursive CFG. The cardinality of the axiom $|S|$ is the cardinality of the universe of derivations $\mathbb{U}$. If the CFG is unambiguous, then $|S|$ also equals the cardinality of the CFG language.

The probability of a derivation $A \overset{+}{\Rightarrow} \gamma, \gamma \in (V \cup \Sigma)^*$, which applies a sequence of production rules $\{r^i\}_{i=1}^n$, denoted by $P(A \overset{+}{\Rightarrow} \gamma, \{r^i\}_{i=1}^n)$, is defined as the probability of iteratively applying each of the n production rules in $A \overset{+}{\Rightarrow} \gamma$, $P(\{r^i\}_{i=1}^n)$. This term is obtained by multiplying the probability of choosing each production rule $r \in R$ in the i^{th} derivation step, $P(r^i)$:

$$P(A \overset{+}{\Rightarrow} \gamma) = P(\{r^i\}_{i=1}^n) = \prod_{i=1}^{n} P(r^i). \tag{1}$$

If $\#R_A$ is the number of production rules in R_A, then $P(r) = 1/\#R_A$, $r \in R_A$, in the case of the CFG is nonstochastic. For stochastic CFG, $P(r)$ is the custom probability assigned to the rule r.

If the CFG is unambiguous, then $P(A \overset{+}{\Rightarrow} \gamma, \{r^i\}_{i=1}^n)$ equals the probability of generating the string γ from A, applying the derivation $\{r^i\}_{i=1}^n$, which is unique. In particular, the probability of producing a sentence $s \in \Sigma^*$ from the axiom S using an unambiguous CFG, $P(S \overset{+}{\Rightarrow} s)$, can also be noted as $P(s)$.

The average probability of any terminal string derivation $A \overset{+}{\Rightarrow} s$, firstly applying the production rule $r : A ::= \gamma$, $\gamma \in (V \cap \Sigma)^*$, equals the probability of firstly choosing r divided by the number of possible terminal string derivations that r can produce:

$$\langle P(A \overset{+}{\Rightarrow} s, r) \rangle = \frac{P(r)}{|r|}. \tag{2}$$

Again, if the CFG is unambiguous, then $\langle P(A \overset{+}{\Rightarrow} s, r) \rangle$ equals the average probability of generating the sentence s from r, which can be noted as $\langle P(s, r) \rangle$.

The GGGP initial population distribution is difficult to control since it strongly depends on the CFG. The production rule cardinalities and probabilities shape the initial distribution. According to (2), the average probability of generating any terminal string s from A depends on the cardinalities of each production rule in $A \overset{+}{\Rightarrow} s$. Let $r_i : A ::= \gamma_i$ and $r_j : A ::= \gamma_j$; $r_i, r_j \in R_A$, be two different production rules. According to (2), if $P(r_i) \approx P(r_j)$ and $|r_i| > |r_j|$, then $\langle P(s, r_i) \rangle < \langle P(s, r_j) \rangle$. It is less likely to obtain, on average, a sentence s starting from r_i than starting from r_j. Therefore, the population initialization is biased to those production rules with lower cardinalities.

According to (1), the probability of a derivation decreases as the number of derivation steps required increases. Since this value is obtained from a product of probabilities, it goes to 0 as the number of production rules in $\{r^i\}_{i=1}^n$ increases. This fact favors the production of smaller derivation trees. Therefore, most of the search space generated by recursive CFG is unreachable during the initialization since large derivations starting from the same recursive production rule are unprovable.

Initialization methods select any production rule with the same nonterminal symbol on the left-hand side with the same probability when using nonstochastic CFG. This setup does not ensure at all population uniformity. Stochastic CFG shape the production rules selection to seek custom distributions. A proper tune of the production rules probabilities may help to grant uniformity in non-recursive CFG. However, manually assigning and tuning these probabilities can be tedious and inaccurate. In the case of recursive CFG, tuning the production rule probabilities is insufficient since these probabilities are static during each derivation tree generation. Let G_{rec} be the right-recursive CFG defined in (3). G_{rec} generates the language comprised of any n-length strings or sentences with the terminal symbol 1: $L(G_{\text{rec}}) = 1^n$.

$$\begin{aligned} &G_{\text{rec}} = (V, \Sigma, R, S) \\ &V = \{S, \mathit{Recursive}\} \\ &\Sigma = \{1\} \\ &R = \{ \\ &\qquad r_1 : \quad S ::= \mathit{Recursive} \\ &\qquad r_2 : \quad \mathit{Recursive} ::= 1\ \mathit{Recursive} \\ &\qquad r_3 : \quad \mathit{Recursive} ::= 1 \\ &\}. \end{aligned} \tag{3}$$

According to (1), $P(S \overset{+}{\Rightarrow} 1^n) = \prod_{i=1}^{n+1} P(r_i)$. Since $\{r_i\}_{i=1}^{n+1} = \{r_1, r_2, ..., r_2, r_3\}$, $P(S \overset{+}{\Rightarrow} 1^n) = P(r_1)P(r_2)^{n-1}P(r_3)$, where $P(r_1) = 1$, $P(r_2) = 0.5$ and $P(r_3) = 0.5$. Therefore, $P(S \overset{+}{\Rightarrow} 1^n) = 0.5^n$. Consequently,

$$\lim_{n \to \infty} P(S \overset{+}{\Rightarrow} 1^n) = 0.$$

For relatively small values of n, the probability of generating the sentence 1^n is already very low. If $n = 8$, then the resulting probability is already $P(S \overset{+}{\Rightarrow} 1^8) = 0.5^8$, 0.0039. This result means a strong bias in the population initialization that tends to generate short sentences. Likewise, almost 50% of the generated derivation trees do not even involve r_2, so the length of their resulting sentences is just 1.

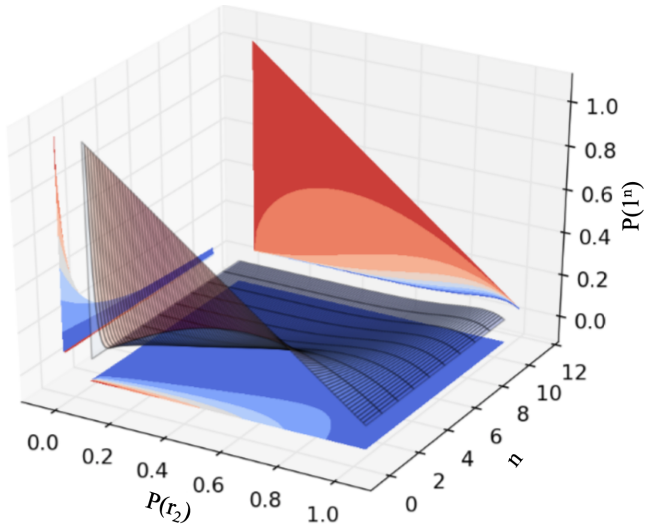


Fig. 1 Probability distribution of obtaining sentences of length n, $P(1^n)$, when the probability of choosing the right-recursive production rule r_2 of G_{rec}, $P(r_2)$, varies. Results reported after generating 1000 individuals using G_{rec}. The projections of the probability distribution in the three planes represent the probability values using a color scale from dark blue for low values to red for probability values close to 1

A stochastic CFG can be applied to bias population initialization to increase the average sentence length. Figure 1 shows the probability of obtaining sentences of length n, $P(1^n)$, when $P(r_2)$ varies, using G_{rec}. Probability distributions are still strongly biased towards short sentences for $P(r_2) < 0.8$. As $P(r_2)$ approaches 1, the probability distributions tend to be uniform, but at the expense of generating very long sentences. These sentences are not suitable for GGGP because they produce code bloat. A usual solution consists of setting a maximum derivation tree depth bound. Such a solution with $P(r_2) \to 1$ and a fixed maximum depth bound biases the individuals towards the maximum size. If $P(r_2) = 0.99$ and the maximum depth bound is set to $n = 10$, then $P(1^{10}) = 0.904$, while the rest of individuals representing shorter sentences sum a probability of 0.096.

3 The grammatically uniform population initialization

The grammatically uniform initialization (GUPI) method generates individuals uniformly distributed in the search space defined by a CFG with a code bloat control mechanism. GUPI grants producing any derivation of a CFG with the same probability. According to (2), the average probability of any terminal string derivation $A \overset{+}{\Rightarrow} s$, applying first the production rule r depends on the probability of applying r, $P(r)$, and its cardinality $|r|$. The cardinality of a production rule may vary during each derivation tree generation. For example, if $A ::= \gamma$ is a recursive production rule, $|A ::= \gamma|$ is not the same at the beginning of the derivation than when it comes to an end due to a depth bound. Therefore, GUPI adjusts the probabilities to the production rules accordingly to give any derivation the same probability of being generated.

Since a nonterminal symbol A produces $|A|$ different terminal string derivations, then $P(A \overset{+}{\Rightarrow} s) = \frac{1}{|A|}$, independently of the starting production rule r applied. Therefore, $\langle P(A \overset{+}{\Rightarrow} s, r)\rangle = \frac{1}{|A|}$, $\forall A \in V$, $\forall r \in R_A$. From (2), $\langle P(A \overset{+}{\Rightarrow} s, r)\rangle = \frac{P(r)}{|r|} = \frac{1}{|A|}$. Therefore, the production rules probabilities must be assigned as:

$$P(r) = \frac{|r|}{|A|}, \tag{4}$$

which is valid for both recursive and non-recursive CFG. The initialization process must include a code bloat control mechanism in case of recursive CFG. Under these conditions, the cardinalities of the production rules and nonterminal symbols may vary during the generation of each derivation as the tree size varies. Therefore, GUPI dynamically recalculates the probabilities of the production rules according to (4) to comply with grammatical uniformity.

GUPI includes a code bloat control mechanism by limiting the number of recursive production rules that can be applied to produce a new individual, instead of the usually employed tree depth bound setting. This recursion bound is set at the beginning of the initialization process. During each individual (tree) generation, GUPI selects either recursive or non-recursive production rules according to their dynamically calculated probabilities while the maximum recursion bound is not reached. Then, it can only choose non-recursive productions until the derivation finishes in a finite number of rewriting steps. Algorithm 1 describes GUPI. It needs three input parameters: the CFG, $G = (V, \Sigma, R, S)$, the population size, N, and the maximum recursion bound, M_ρ, to produce a grammatically uniform population.

Let us suppose an example of generating just an individual ($N = 1$) using Algorithm 1 with G_{rec} (3) and $M_\rho = 2$. Therefore, the first loop *for* calls the function *generate_individual*$(S, 2)$ of line 4. S is the axiom of the grammar. The condition does not comply in line 5 since R_S has the production rule r_1, $R_S = \{r_1 : S ::= \mathit{Recursive}\}$. According to Formula (4), $P(r_1) = 1$, since $|r_1| = 3$ and $|S| = 3$, which means that three different derivations can be generated from S or r_1 with $\rho = 2$: $\{S ::= \mathit{Recursive}; \mathit{Recursive} ::= 1\}$, $\{S ::= \mathit{Recursive}; \mathit{Recursive} ::= 1\mathit{Recursive}; \mathit{Recursive} ::= 1\}$, $\{S ::=$

Algorithm 1 Grammatically uniform population initialization, GUPI. Inputs: G, N, M_ρ

1: $population = \emptyset$
2: **for** k in range (1,N) **do** ▷ Generate N individuals
3: $population \cup generate\_individual(S, M_\rho)$
return *population*
4: **function** *generate_individual*(nonterminal A, int ρ)
5: **if** $R_A = \emptyset$ **then** ▷ The stop condition
6: **return** $\emptyset$
7: Get the set of production rules $R_A = \{r_i : A :: \gamma_i\}$ and compute their probabilities $\{P(r_i)\}$ according to Formula 4 and ρ ▷ Note that If ρ is 0 and r_1 is recursive, then $P(r_i) = 0$
8: Select a production rule $r : A ::= \gamma \in R_A$ following the computed probabilities $\{P(r_i)\}$
9: **if** $\gamma \cap V = \emptyset$ **then** ▷ γ has not nonterminal symbols
10: **return** $r \cup generate\_individual(\emptyset, \rho)$
11: **if** $A \in \gamma$ **then** ▷ r is recursive
12: $\rho \leftarrow \rho - 1$
13: **for all** nonterminal $B_i \in \gamma$ **do**
14: **if** B_i does not lead to recursive derivations **then**
15: **return** $r \cup generate\_individual(B_i, 0)$
16: **else**
17: Assign ρ_i a random value, such that $\sum_i \rho_i = \rho$
18: $r \cup generate\_individual(B_i, \rho_i)$

Recursive; *Recursive* ::= 1*Recursive*; *Recursive* ::= 1*Recursive*; *Recursive* ::= 1}.

Line 8 selects $r_1 : S ::= Recursive$, the only production rule in R_S with a probability of 1, $P(r_1) = 1$. r_1 satisfies neither conditions of line 9 nor line 11. From line 13, $B_1 = Recursive$, the only nonterminal symbol in the right-hand side of r_1. Since *Recursive* can lead to a recursive production rule, the algorithm adds r_1 to the current individual and makes a recursive call to $generate\_individual(Recursive, 2)$.

At this point, the current individual is the derivation $\{S ::= Recursive\}$; $A = Recursive$ and $\rho = 2$. In this second execution of the function *generate_individual*, the algorithm calculates $R_{Recursive} = \{r_2 : Recursive ::= 1Recursive;\ r_3 : Recursive ::= 1\}$ on line 7. Also, $\{P(r_2) = 2/3, P(r_3) = 1/3\}$ since $|Recursive| = 3$, $|r_2| = 2$ and $|r_3| = 1$ considering $\rho = 2$. Note that all possible derivations that the initialization process can generate are equiprobable. Let's suppose that line 8 selects r_2. Then, $\rho = 1$ (line 12) and $B_1 = Recursive$, which leads to recursive derivations. Line 17 makes $\rho_1 = 1$, and line 18 adds r_2 to the current individual, calling $generate\_individual(Recursive, 1)$ afterward. The current derivation is now $\{S ::= Recursive; Recursive ::= 1Recursive\}$.

The line 7 of this third iteration calculates $R_{Recursive} = \{r_2 : Recursive ::= 1Recursive; r_3 : Recursive ::= 1\}$, but in this case, $\{P(r_2) = 1/2, P(r_3) = 1/2\}$ since $|Recursive| = 2$, $|r_2| = 1$, and $|r_3| = 1$. Note that the proposed algorithm assigns the probabilities to the production rules dynamically, depending on the context of the individual under creation, particularly ρ. Let's suppose the algorithm selects r_2 again on line 8. Then, $\rho = 0$ and $B_1 = Recursive$. Since *Recursive* leads to recursive derivations, $\rho_1 = 0$, and line 18 adds r_1 to the current individual and calls the function $generate\_individuals(Recursive, 0)$. At this moment, the current individual is $\{S ::= Recursive; Recursive ::= 1Recursive; Recursive ::= 1Recursive\}$.

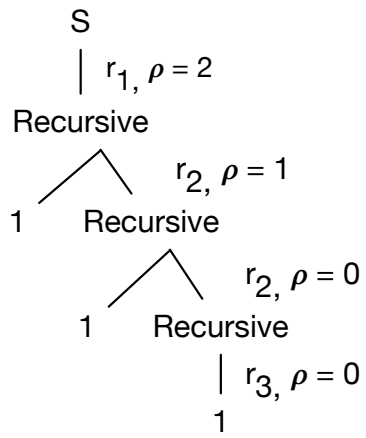


Fig. 2 A sample profound derivation tree generated by GUPI with G_{rec}. It is equally probable as any other in the search space.This derivation represent the terminal string 111

During the fourth iteration of the individual creation, $R_{Recursive} = \{r_2 : Recursive ::= 1Recursive; r_3 : Recursive ::= 1\}$, but $P(r_2) = 0$ since $|r_2| = 0$ because of $\rho = 0$. r_2 cannot generate terminal string derivations because it is a recursive production rule, and the maximum recursion bound has been reached. As $P(r_3) = 1$, the algorithm can only select this production rule on line 8. The right-hand side of r_3, 1, meets the condition of line 9. Therefore, line 10 adds r_3 to the current derivation and invokes the function $generate\_individual(\emptyset, 0)$. Then, the function stops since the condition of line 5 is met, and the derivation generated is added to *population*, shown in Figure 2 as a derivation tree. The *for* loop of line 2 would call the function $generate\_individual(S, 2)$ again to generate another individual for the initial population if k were not yet N, which is not the case of the current example. Finally, the algorithm stops returning the initial population distributed uniformly.

4 Results

Four different problems are proposed to compare both the initial population distribution and the performance

of the overall GGGP evolutionary process when us-ing either the proposed GUPI and a standard initial-ization algorithm with code bloat control for GGGP, which has reported excellent results in comparison to other significant genetic programming and GGGP initialization algorithms (García Arnau et al., 2007). This standard initialization process consists of generating derivations by choosing any production rule (at random) from a nonterminal symbol with the condition of not surpassing a preset maximum number of applications of recursive production rules (a recursion bound). The use of stochastic CFG has not been considered in the experiments because of its poor results. The assignation of high probability values to recursive production rules to pursue uniformity biases the population toward larger derivation trees, which negatively affects the subsequent evolutionary process.

The four problems are noted by the CFG involved: G_{rec}, $G_{\text{MLP}}(I,O)$, G_{RBS}, and G_{SR}. The first three problems are employed to conduct experiments to reveal whether or not GUPI approximates uniform initial populations, whereas $G_{\text{MLP}}(I,O)$, G_{RBS}, and G_{SR} serve to compare the performance of the overall evolutionary process that takes place after the initialization stage. Table 1 summarizes the features of each grammar and the study in which they are involved.

$$
\begin{aligned}
&G_{\text{MLP}}(I,O) = (V, \Sigma, R_{I,O}, MLP)\\
&V = \{MLP, N, D, C, A, Z, B\}\\
&\Sigma = \{0, 1\}\\
&R_{I,O} = \{\\
&\quad r_1: \quad MLP ::= N \mid D\\
&\quad r_2: \quad D ::= B^{IO}\\
&\quad r_3: \quad N ::= N\ C \mid C\\
&\quad r_4: \quad C ::= A\ Z\\
&\quad r_5: \quad B ::= 0 \mid 1\\
&\quad r_6: \quad A ::= 1B^{I-1} \mid B1B^{I-2} \mid\\
&\qquad\quad B^j1B^{I-1-j} \mid ... \mid B^{I-2}1B \mid B^{I-1}1\\
&\quad r_7: \quad Z ::= 1B^{O-1} \mid B1B^{O-2} \mid\\
&\qquad\quad B^j1B^{O-1-j} \mid ... \mid B^{O-2}1B \mid B^{O-1}1\\
&\}.
\end{aligned}
\tag{5}
$$

G_{rec}, defined in Formula (3) and taken as an example to illustrate GUPI, is also proposed as the first problem to study the initial population distributions. This is a small recursive CFG that generates strings with the nonterminal symbol 1 of any length via the recursive production rule r_2 that permits to observe the behavior of the initialization process. $G_{\text{MLP}}(I,O)$, defined in Formula (5), is employed to compare the performance of the overall GGGP evolutionary process when starting by either GUPI or the standard approach. It is a parametrized CFG that encodes feedforward one-hidden-layer artificial neural networks with I input neurons, O output neurons, and any number of partially-connected hidden neurons represented by the nonterminal symbols N and C (Couchet et al., 2007). This CFG contains the recursive production rule r_3 to generate any number of hidden neurons. Therefore, it is structurally similar to G_{rec} but closer to a real application of an evolutionary algorithm, which makes it more adequate for performance tests.

$$
\begin{aligned}
&G_{\text{RBS}} = (V, \Sigma, R, RBS)\\
&V = \{RBS, R, A, C, CL, EV, OP, VL, EQ, VA\}\\
&\Sigma = \{if, then, =, !=, \&, \|, v1, v2, v3, v4, t, f\}\\
&R = \{\\
&\quad r_1: \quad RBS ::= R\ RBS \mid R\\
&\quad r_2: \quad R ::= if\ A\ then\ C\\
&\quad r_3: \quad A ::= CL\ A \mid CL\\
&\quad r_4: \quad CL ::= EV\ OP\ EV\\
&\quad r_5: \quad EV ::= VA\ EQ\ VL\\
&\quad r_6: \quad VA ::= v1 \mid v2 \mid v3 \mid v4\\
&\quad r_7: \quad EQ ::= != \mid =\\
&\quad r_8: \quad OP ::= \| \mid \&\\
&\quad r_9: \quad VL ::= t \mid f\\
&\quad r_{10}: \quad C ::= v1 \mid v2 \mid v3 \mid v4\\
&\}.
\end{aligned}
\tag{6}
$$

$$
\begin{aligned}
&G_{\text{SR}} = (V, \Sigma, R, SR)\\
&V = \{SR, E, OP1, OP2\}\\
&\Sigma = \{+, -, /, *, sin, cos, v\}\\
&R = \{\\
&\quad r_1: \quad SR ::= E\\
&\quad r_2: \quad E ::= OP1\ E\\
&\quad r_3: \quad E ::= OP2\ E\ E\\
&\quad r_4: \quad E ::= v\\
&\quad r_5: \quad OP1 ::= sin \mid cos\\
&\quad r_6: \quad OP2 ::= + \mid -\\
&\quad r_7: \quad OP2 ::= / \mid *\\
&\}.
\end{aligned}
\tag{7}
$$

Formulas (6) and (7) define the other two CFG employed in both comparison studies: the initial population distribution and the performance of the evolutionary process. G_{RBS} encodes knowledge bases of rules.

Table 1 CFG involved in the experiments. In parenthesis, the reference to the definition of each grammar

CFG	Features	Study
G_{rec} (3)	One recursive production rule	Initial distribution
$G_{\text{MLP}}(I, O)$ (5)	Real-world problem with one recursive production rule	Performance
G_{RBS} (6)	Two recursive production rules	Initial distribution and performance
G_{SR} (7)	One recursive and one double recursive production rules	Initial distribution and performance

It contains two recursive production rules: r_1 to determine the number of rules included in the knowledge base and r_3 for the number of clauses in the antecedent of each rule. Finally, G_{SR} defines the expressions of a symbolic regression optimization problem with two recursive productions, r_2 and r_3, to create unary and binary operations, respectively. This latter CFG presents a further difficulty: r_3 is double recursive since E yields itself twice.

4.1 Initial population distribution

Six experiments have been conducted for G_{rec}, G_{RBS}, and G_{SR} to show the distribution of the individuals generated by GUPI and the GGGP standard initialization algorithm in terms of the number of recursive productions applied. These experiments correspond to the maximum recursion bounds 5, 10, 20, 50, 100, and 500, noted as M_ρ in the case of GUPI (see Algorithm 1). Each experiment consisted of generating 1000 individuals by each of the two initialization approaches.

Figure 3 shows the most representative results, comparing the distribution of the individuals generated by GUPI, in blue, and the standard initialization algorithm, in orange. This figure includes six pairs of histograms out of a total of eighteen, where the abscissa axes represent the number of recursive derivations applied by either GUPI or the standard initialization algorithm to produce a derivation tree. The ordinate axes show the absolute frequency of derivation trees generated for each abscissa value. Figures 3(a) and 3(b) correspond to individuals generated using G_{rec}; Figures 3(c) and 3(d) correspond to G_{RBS}; and, finally, Figures 3(e) and 3(f) for G_{SR}. The missing histograms have similar shapes.

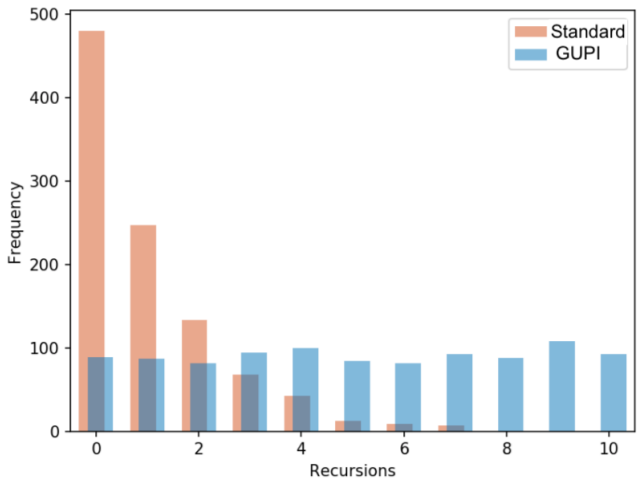
From figure 3(a), it is clear that the standard population initialization approach already shows a bias towards derivation trees with few recursive production rules even for the lowest recursion bound (10). Note that the standard approach shows a limit at 7 recursive productions, and almost no derivation tree exceeds it. Conversely, GUPI approximates a uniform distribution of individuals in terms of the number of recursive production rules applied. This scenario is similar in figure 3(b), where the recursion bound raises to 20, and the same for 50, 100, and 500 (not shown). GUPI still approximates a uniform population for high recursion bounds.

Figures 3(c) to 3(f) correspond to CFG with two recursive production rules, G_{RBS}, and G_{SR}. In these cases, the standard initialization approach reduces its bias towards derivation trees with few recursive productions. However, it is still strongly biased to derivations with less than 13 recursive productions in the case of Figures 3(c) and 3(d) or less than 25 in the case of Figures 3(e) and 3(f). It is noteworthy that the derivation trees generated by the standard initialization approach with G_{SR} can be larger than those using G_{RBS} because the double recursive production rule r_3 in G_{SR} increases the probability of generating derivation trees with more recursive derivations. On the contrary, GUPI always approximates a uniform distribution of recursive productions applied even for the highest recursion bounds, Figure 3(f), which is essential in these kinds of problems where the solutions usually involve many rules or operations.

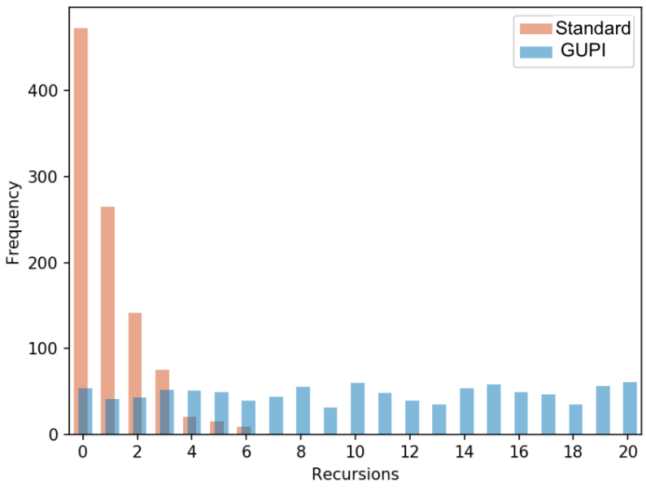
4.2 Performance

Six experiments have been conducted for $G_{\text{MLP}}(I, O)$, G_{RBS}, and G_{SR} to show the performance of the overall grammar-guided evolutionary process regarding convergence speed and quality of the solutions achieved, measured by their fitness. The experiments consisted of trying to obtain derivation trees with 5, 10, 20, 50, 100, and 250 recursive production rules from initial populations generated by GUPI and the standard GGGP initialization approach. Two GGGP approaches have been considered, employing either a generational replacement estimation of distribution algorithm (EDA) (Kim et al., 2014) or the Whigham's crossover (WX) (Whigham, 1995) with a steady-state strategy as variation operators. The aim is to observe the evolutionary algorithm starting from GUPI and the standard approach using variation operators with very different behaviors.

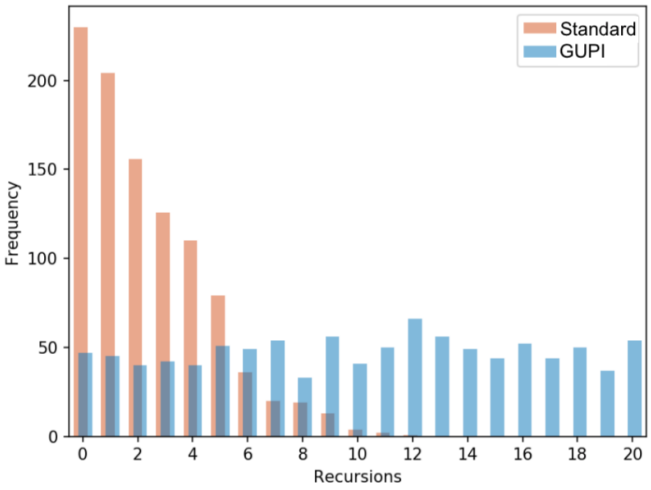
An estimation of distribution algorithm learns a probabilistic model that shapes the distribution of the individuals in the current population. Then, the EDA re-



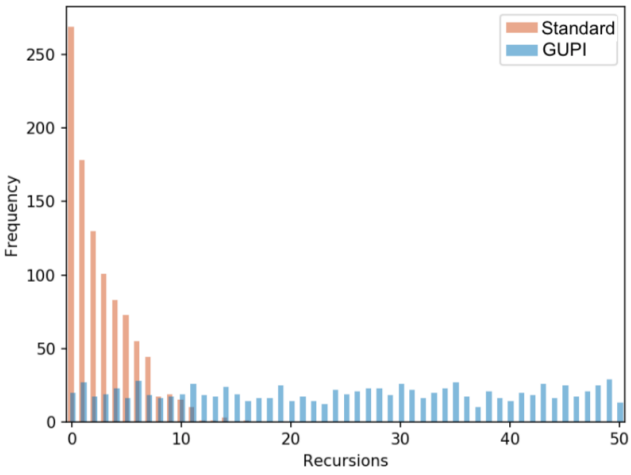
(a) Recursion bound: 10. G_{rec}



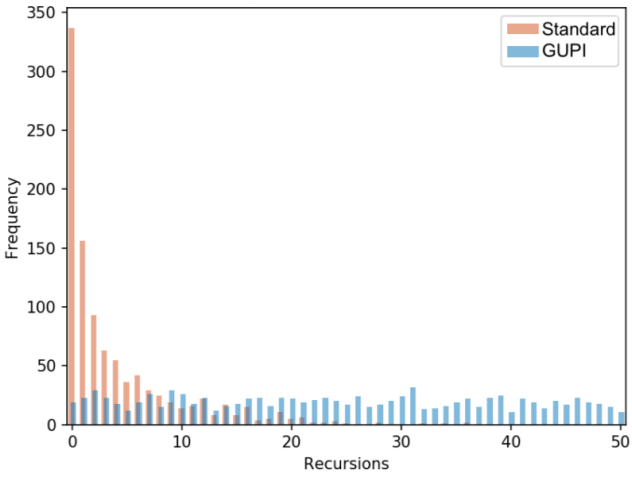
(b) Recursion bound: 20. G_{rec}



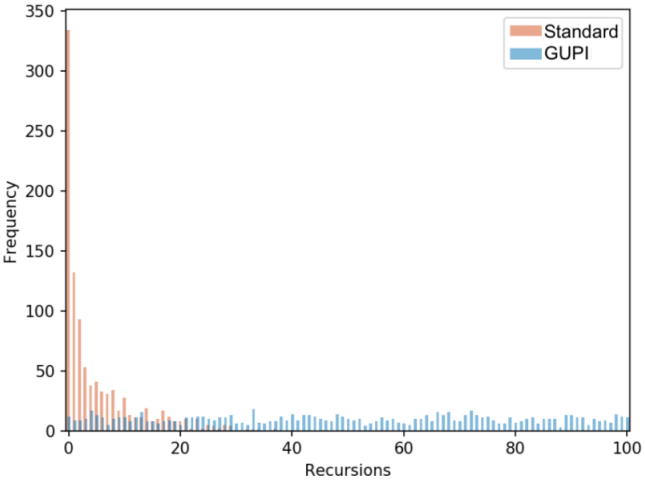
(c) Recursion bound: 20. G_{RBS}



(d) Recursion bound: 50. G_{RBS}



(e) Recursion bound: 50. G_{SR}



(f) Recursion bound: 100. G_{SR}

Fig. 3 Most representative results comparing the distribution of 1000 individuals generated by GUPI (blue) and the standard initialization algorithm (orange) for different recursion bounds for the three CFG under study: G_{rec}, G_{RBS}, and G_{SR}

places part of or the entire current population by a set of individuals generated according to the probabilistic model previously learned in each generation to drive the evolutionary process towards promising individuals. EDA improve GGGP performance by boosting its local search capability, although there also exists the risk of converging to local optima due to the loss of genotypic diversity (Kim and McKay, 2013).

WX is a very often and widely tested variation operator used in GGGP that, in general, achieves satisfactory results. It selects two parent individuals (derivation trees) from the current population through the selection operator. Then, it randomly chooses a (crossover) node from each parent with the same nonterminal symbol, and finally generates the offspring by swapping the parent subtrees from the crossover nodes.

Each experiment consists of 50 executions with three steps each. The first step generates a target derivation tree following a uniform distribution containing precisely the number of recursive production rules established for the experiment. For example, in the experiment with 5 recursive production rules for the G_{RBS} problem, which contains two recursive productions, the target derivation tree generated at each execution has either 5 recursions of the nonterminal RBS, A or both with exactly 5 recursions in total. The second step creates the initial population using GUPI and the standard approach for comparative purposes. Finally, the third step executes the GGGP evolutionary process from both initial populations using EDA and WX as variation operators to obtain the target derivation tree or a similar one. Therefore each execution performs four runs: GUPI+EDA, GUPI+WX, standard+EDA, and standard+WX.

The GGGP system employed in the experiments is the same except for the variation operator: EDA or WX. The initial population size is 100 individuals, generated by GUPI and the standard initialization to compare both approaches in terms of convergence speed and quality of the solutions achieved (fitness) by the GGGP system using either EDA or WX. WX employs two parent individuals chosen by applying twice the tournament selection operator of size two to produce an offspring of two new individuals. Then, a steady-state replacement strategy is adopted, replacing the two worst individuals of the current population by the off-spring. This strategy tends to explore the search space widely, which avoids local optima but slows down the convergence speed.

EDA starts from 50 parent individuals, selected from the population by applying the tournament selection method of size two, to calculate the probabilistic model from which it creates a new whole generation. The probabilistic model consists of assigning a probability to each production rule in the CFG. These probabilities are calculated from the absolute frequencies of the ap-plied production rules to generate the parent individu-als. The derivation trees of the offspring are then gen-erated by choosing production rules according to the probabilities previously calculated that yield any sentence or terminal string s of the CFG from the axiom S (Ramos Criado, 2017). Contrary to WX, EDA adopts a generational replacement model that substitutes the whole current population by individuals generated from the previously calculated probabilistic model. This approach boosts the local search of the evolutionary algorithm, which increases the convergence speed but, in turn, likely leads the population to a premature convergence to a local optimum.

Both WX and EDA GGGP systems do not apply either mutation or immigration to reduce the random component of the evolutionary process and thus, facilitate the comparisons. The evolutionary process stops when it finds the target derivation tree, or the average population fitness does not improve after 50 generations.

Since the GGGP searching problem for these experiments focuses on obtaining the target derivation tree or a similar one, the fitness function compares the sentence encoded in the target derivation tree with the solution encoded by the individual to evaluate based on the definition of the Hamming distance (Roth, 2006).

Let $s = a_1 a_2 \cdots a_n$ and $s' = a'_1 a'_2 \cdots a'_n$ be two strings of equal length n, the Hamming distance be-tween them is the number of positions at which the corresponding symbols a_i and a'_i are different. Then, the fitness value is the Hamming distance between the sentence encoded in the target derivation tree s and the solution to evaluate s' when both are of the same length n. Since these two sentences might not be of the same length, the proposed fitness function extends the defi-nition of the Hamming distance to cover this scenario by summing their difference in length to the previously calculated fitness value. Therefore, the definition of the fitness function $f_s(s')$, given the target $s = a_1 a_2 \cdots a_n$
and the solution to evaluate $s' = a'_1 a'_2 \cdots a'_m$, is the following:

$$f_s(s') = \sum_{i=1}^{\max\{n,m\}} d(a_i, a'_i), \tag{8}$$

where

$$d(a_i, a'_i) = \begin{cases} 1, & \text{if } a_i \neq a'_i \text{ or } i > \min\{n, m\}, \\ 0, & \text{otherwise.} \end{cases}$$

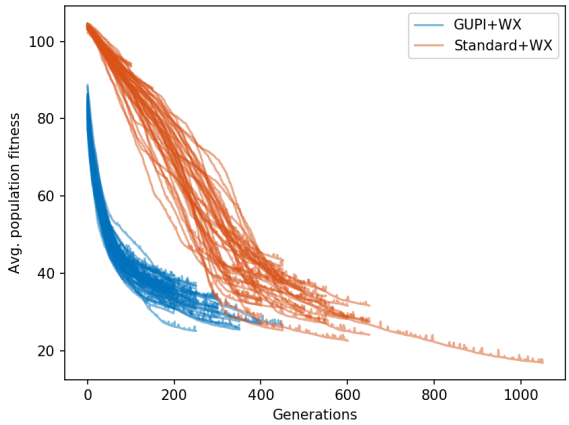
The evolutionary process seeks to minimize the fitness, with an optimum value of zero. Therefore, the experiments are not interested in the semantics of each problem, as the search for the best knowledge base of rules that solves a specific task in the case of G_{RBS}. This definition of the experiments avoids the noise or bias produced by further (optimization) processes needed to make the system work in a particular environment. For example, learning, adjusting or tuning the specific values corresponding to the terminals $v1$, $v2$, $v3$, and $v4$ in G_{RBS}.

Two types of plots present the results achieved by the experiments performed in each of the three problems under study. Evolution charts of the average population fitness compare the convergence speed of both GGGP systems (WX and EDA) when starting from a population initialized by GUPI or the standard approach (see Figures 4(a), 4(b), 5(a), 5(b), 6(a), and 6(b)). The abscissa axis of the evolution charts represents the GGGP generation against the average population fitness value, represented by the ordinate axis. Blue-colored lines represent the fitness evolution starting by a GUPI initialized population, while the orange series correspond to the standard initialization approach. There are fifty evolution lines for each initialization approach matching the 50 executions run for each experiment. They start from generation 0 (initial population) and finish when the GGGP evolution stops. Since there are a total of twelve pairs of plots (comparing GUPI and the standard approach), one per target derivation tree size and variation operator, the figures show the most representative experiments run.

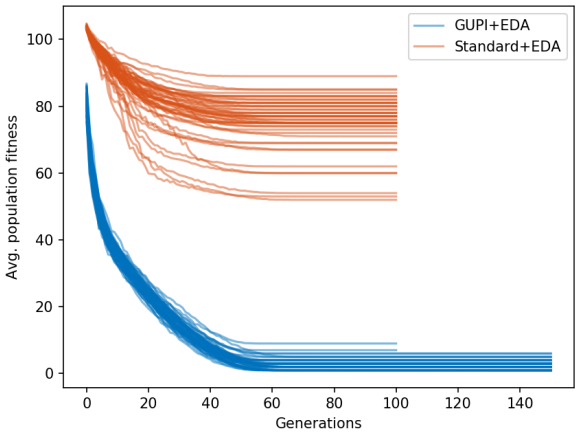
The second kind of plots comprises two different charts that show statistical results to compare the fitness of the solutions achieved by GUPI+WX versus standard+WX, and GUPI+EDA versus standard+EDA for each target derivation tree in each of the three problems under study (Figures 4(c), 4(d), 5(c), 5(d), 6(c), and 6(d)). 50 executions have been run for each scenario. Both charts share the abscissa axis, which represents the number of recursive production rules of the target derivation tree. The chart below shows six pairs of box-and-whisker plots representing the fitness of the best solutions achieved by the GGGP system (with WX or EDA) starting from GUPI (in blue) and the standard initialization approach (in orange) for each of the six target derivation tree sizes.

Statistical comparisons have been conducted via a one-way analysis of variances (ANOVA) on two groups (t-test) of the average fitness of the best individuals achieved in each of the 50 executions run for each experiment. The chart above points to the significance level achieved by the ANOVA tests. The dependent variable is the best fitness achieved, and the independent one is the initialization approach employed to run the experiments. The aim is to study the impact of the initial population on the fitness of the final solutions achieved by the evolutionary process by gathering empirical evidence of whether the differences between the means of the dependent variable are statistically significant. A significance level of $p < 0.05$ rejects the null hypothesis (the means are equal). The plot above points out this significant-level threshold by a dashed line. A necessary condition for ANOVA tests is that the variances within groups of the independent variable are equivalent. However, ANOVA is robust to this violation when the groups are of an equal or near size, which is satisfied since each experiment consists of 50 executions (samples) for each initialization approach and variation operator.

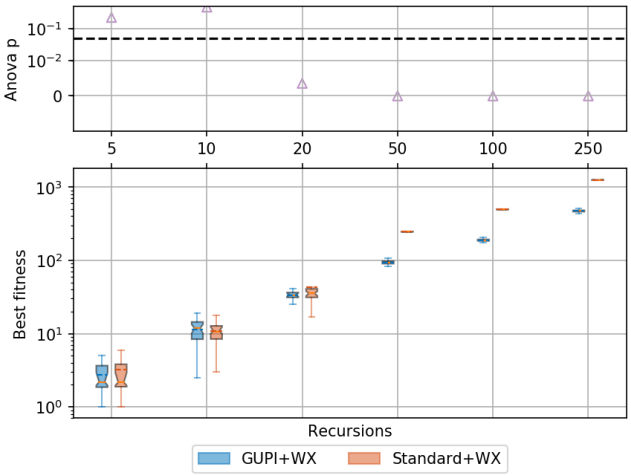
Figure 4 shows the performance achieved by the experiments using $G_{\mathrm{MLP}}(I, O)$. These results are independent of the parameters I and O since they correspond to the input and output neurons respectively and do not affect the size of the target derivation trees (the number of recursive production rules). Nevertheless, these parameters need to be set to instantiate the grammar and run the experiments, so $I = 4$ and $O = 2$ have been chosen, noted as $G_{\mathrm{MLP}}(4, 2)$. Figures 4(a) and 4(b) show the convergence speed of the GGGP system using WX and EDA variation operators, respectively, searching for a target derivation trees with 20 production rules. These are the two most representative evolution charts for the $G_{\mathrm{MLP}}(4, 2)$ problem. These two figures evidence the difficulties of the evolutionary process when using the standard population initialization or, in general, any initialization algorithm that can not generate populations spread enough in the search space. Moreover, Figure 4(b) clearly shows the premature convergence of standard+EDA. Since EDA performs local search better than WX, it also converges faster, but to inadequate solutions when the initial population is not uniformly distributed. For a higher level of recursions, the evolutionary process starting from a population generated by GUPI is always faster than using the standard approach. On the contrary, the results also report that the population initialization barely affects the evolutionary process when the target derivation trees have 5 or 10 recursive productions. For such small derivation trees, both population initialization approaches produce a suitable representative sample of the search space — Review Figure 3(a). Although GUPI approximates better than the standard approach a uniform distribution, the latter can still reach derivation trees with 7 recursive productions.



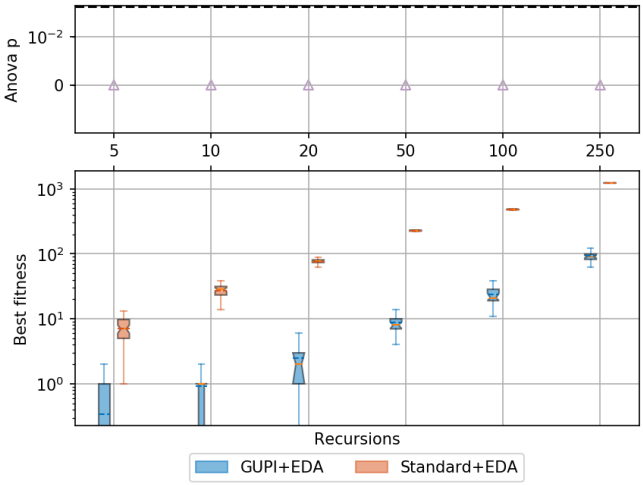
(a) WX. Target derivation trees with 20 recursions.



(b) EDA. Target derivation trees with 20 recursions.



(c) Statistical comparisons using WX as the GGGP variation operator.



(d) Statistical comparisons using EDA as the GGGP variation operator.

Fig. 4 $G_{\text{MLP}}(4, 2)$ problem. 50 executions run for each initialization algorithm, variation operator, and target derivation tree size (recursions). Above, the evolution of the average population fitness using either WX or EDA when starting from a population initialized by GUPI (blue) and the standard initialization algorithm (orange). Below, box-and-whisker plots and significance levels p achieved by the t-tests performed to compare GUPI and the standard initialization approach in terms of the fitness of the final solutions achieved by the evolutionary process starting from initial populations generated by GUPI and the standard approach. The dashed line points out the significance-level threshold that rejects the means are equal

Figures 4(c) and 4(d) show that the fitness of the final solutions achieved by GGGP starting from GUPI statistically outscores those achieved by commencing with the usual approach. Figure 4(c) points out two exceptions for this general rule: the experiments conducted to search for target derivation trees with 5 and 10 recursions using WX as the variation operator for the GGGP system. The null hypothesis cannot be rejected in both cases since the significance level reported by the t-tests exceed the preset 0.05 threshold (note the triangles above the dashed line). Therefore, starting by GUPI or the standard approach, there are no differences in the quality of the solutions achieved when searching for small solutions using the WX crossover operator. The reason is that WX performs a search more explorative than EDA, which balances the possible lack of uniformity in some executions of the standard initial population.

Table 2 shows descriptive statistics for the $G_{\text{MLP}}(4, 2)$ problem regarding the fitness of the final solutions achieved by each initialization+variation approach (rows) after 50 executions run for each target derivation tree size (columns). Each execution consists of a GGGP approach searching for a specific target derivation tree of a size

Table 2 Descriptive statistics for the $G_{\text{MLP}}(4, 2)$ problem regarding the fitness of the final solutions achieved by each initialization+variation GGGP approach (rows) after 50 executions run for each target derivation tree size, represented by columns labeled from 5 to 250 recursions. In bold, the means with significantly lower values for each column and the same variation operator

Statistic	Init.+Variation	Recursions					
		5	10	20	50	100	250
Mean	GUPI+WX	2.74	11.40	**33.36**	**93.77**	**188.24**	**471.75**
	Standard+WX	3.19	10.80	42.80	246.18	496.03	1245.81
	GUPI+EDA	**0.34**	**0.92**	**2.52**	**8.82**	**23.82**	**92.56**
	Standard+EDA	7.00	27.34	75.02	226.94	478.96	1227.84
Std. Dev.	GUPI+WX	1.41	4.31	4.00	5.94	6.85	17.42
	Standard+WX	2.23	4.25	22.00	1.38	1.77	2.08
	GUPI+EDA	0.52	1.18	1.90	3.37	8.22	16.10
	Standard+EDA	3.26	6.53	8.29	5.15	6.82	6.00
Min.	GUPI+WX	1.00	2.52	25.13	81.87	175.39	425.26
	Standard+WX	1.00	1.05	16.91	242.64	491.86	1238.69
	GUPI+EDA	0.00	0.00	0.00	4.00	11.00	63.00
	Standard+EDA	1.00	10.00	52.00	213.00	458.00	1205.00
95% Conf. Int.	GUPI+WX	[2.35, 3.13]	[10.21, 12,60]	[32.25, 34.47]	[92.12, 95.41]	[186.3, 190.1]	[466.9, 476.6]
	Standard+WX	[2.57, 3.81]	[9.62, 11.98]	[36.71, 48.90]	[245.8, 246.6]	[495.6, 496.6]	[1245, 1246]
	GUPI+EDA	[0.20, 0.48]	[0.59, 1.25]	[1.99, 3.05]	[7.89, 9.75]	[21.54, 26.10]	[88.10, 97.02]
	Standard+EDA	[6.10, 7.90]	[25.53, 29.15]	[72.72, 77.32]	[225.6, 228,4]	[477.1, 480.9]	[1226, 1230]
Max.	GUPI+WX	6.69	18.94	41.08	107.50	203.41	507.91
	Standard+WX	10.42	20.88	94.22	248.80	499.65	1249.51
	GUPI+EDA	2.00	6.00	9.00	17.00	52.00	150.00
	Standard+EDA	13.00	38.00	89.00	235.00	487.00	1237.00

ranging from 5 to 250 recursions. This table corresponds to the results graphically reported in Figures 4(c) and 4(d). The rows represent the mean, standard devia-tion, minimum, the 95% confidence interval, and maxi-mum values of the 50 best fitness values (final solutions) achieved by GUPI+WX, Standard+WX, GUPI+EDA, and Standard+EDA. Note that the lower the statistic, the better since this is a minimization problem. The Table groups the four values of each sample statistic to ease the comparison between the initialization methods under the two evolutionary conditions: WX and EDA. The means that report statistically significant lower final fitness values for the same variation operator and target derivation tree size are marked in bold.

Except for the case of searching for target derivation trees of size 10 recursions using Standard+WX, both GUPI-initialized evolutionary approaches report lower means of the final fitness values than those us-ing the standard initialization algorithm. Only GGGP approaches initialized by GUPI are marked in bold, meaning that they achieve significantly better solutions than those reached when the standard approach generates the initial population. The differences regarding the mean between GUPI and the standard initialization approaches for the same variation operator increase as it does the size of the target derivation trees. In fact, the differences are statistically significant when com-paring GUPI+WX and Standard+WX for target tree sizes of 20 or more recursions, achieving lower values for GUPI. The same is true in the case of the EDA variation operator for all target derivation tree sizes. The differences between means for the same variation operator are rather substantial when the target derivation tree has 50 or more recursive productions. Moreover, the GUPI+EDA approach reaches values for the mean that are at least 10 times lower than those obtained by the Standard+EDA one. GUPI+EDA performs significantly better than any other combination of initialization method plus variation operator in this problem. Besides, it is the only approach capable of finding the target derivation trees for 5, 10, and 20 recursions as the row corresponding to the minimum fitness values reports, which is also indicative of the great difficulty of the experiments proposed for this problem.

The G_{RBS} problem is more complicated than G_{rec} since the former includes two recursive production rules: r_1 and r_3. Figure 5 presents the results achieved. In this case, the experiments performed show that GUPI outperforms the standard approach in terms of convergence speed of the GGGP evolution process when searching for target derivation trees larger than 10 recursive productions. If the problem gets deeper, for those experiments with target derivations trees of 50 recursions or more (see Figures 5(a) and 5(b) as representative examples), then, unlike GUPI, the GGGP evolution starting with the usual approach prematurely converges.

Again, this effect is due to the standard initialization difficulties to produce representative samples that lead the evolutionary process to promising solutions. On the contrary, GUPI overcomes these difficulties, generating initial populations that evolve faster. Moreover, figures 5(c) and 5(d) show that starting from initial populations generated with GUPI, the solutions achieved by the evolutionary process are statistically better than those produced when the standard approach initializes the population. Just in the case of searching for target derivation trees with 5 recursive productions, Figure 5(d) reveals that the fitnesses of the final solutions are similar. This result supports that the initial population has an insignificant influence on the overall evolutionary process when dealing with small derivation trees. Therefore, GUPI becomes especially interesting for problems that involve large derivation trees.

Table 3, similar to the previous one, shows descriptive statistics about this problem. As for the $G_{\mathrm{MLP}}(4, 2)$ problem, both variation operators usually achieve lower mean values using the GUPI initialization than the standard approach. These differences are substantial when the target derivation tree has 50 or more recursive productions. The only exception is that Standard+WX performs significantly better than initializing the population with GUPI to search for target derivation trees with 5 recursions. This fact reveals that the uniform initialization with GUPI causes the subsequent evolutionary process, whether more explorative (WX) or more exploitative (EDA), to find better solutions than us-ing random (though not uniform) initialization. These differences are more prominent the deeper the target derivation tree to be found. Once again, the GUPI+EDA approach achieves the best results. This time, both GUPI and standard initialization approaches using EDA can find the target derivation tree, but only in the case of size 5 recursions. Since G_{RBS} contains two recursive production rules, it is a hard problem to solve.

G_{SR} also has two recursive production rules (r_2 and r_3), with the added feature that r_3 is double recursive. In this case, the GGGP convergence speed searching for target derivation trees with 5, 10, or 20 recursive derivations is similar when starting from initial populations generated by either GUPI or the standard approach for both WX and EDA variation operators. These results are due to the doubly recursive production rule that can increase the probability of generating deeper derivation trees with the standard initialization algorithm, as seen in Figures 3(e) and 3(f). Therefore, the initial populations approximate a uniform distribution when the recursion bound is less than 20, even for the standard approach. However, when the target derivation trees reach 50 recursive derivations, GUPI-initialized populations always outscore the standard approach in terms of the GGGP convergence speed. Figures 6(a) and 6(b) show that the GGGP evolutionary process started from initial populations generated by the standard approach prematurely converges to local optima for target derivation trees larger than 100.

Figures 6(c) and 6(d) show the results of the t-tests performed using WX and EDA, respectively, to compare the quality of the final solutions provided by the GGGG algorithm starting from initial populations generated by GUPI and the usual approach. Since the standard initialization method with G_{SR} can likely produce derivation trees up to 25 recursive productions, the final solutions achieved by the subsequent evolutionary algorithm have statistically similar fitness than starting from GUPI when searching for the smallest target derivation trees. In particular, in the cases of 5 and 20 recursive productions using WX as variation operator, and 5 recursions when using EDA. For larger target derivation trees, the evolutionary algorithm starting from initial populations generated by GUPI always outperforms the standard approach significantly using either variation operator in terms of the fitness of the final solutions achieved.

Table 4 shows descriptive statistics regarding the fitness of the final solutions achieved when solving the G_{SR} problem. GUPI with either WX or EDA reaches lower values of the mean than the standard initialization approach for most of the experiments conducted. However, the differences of the mean are not so strik-ing as reported in the two problems above, $G_{\mathrm{MLP}}(4, 2)$ and G_{RBS}. They are statistically significant for target derivation trees of size 10 and 50 recursions or more when comparing GUPI+WX and Standard+WX, being GUPI better that the standard initialization method (see the mean values in bold). The general rule observed in the previous problems also applies in this case when comparing the fitness of the final solutions achieved by the EDA variation operator starting from either initialization approach: the differences of the mean values become significant for 10 or more recursions, being GUPI always better than the standard approach. Again, it is essential to notice that the larger the target derivation tree to be found, the higher is the difference observed between the values of the mean. Unlike the other two problems, the GUPI+EDA combination does not show notably better performance compared to the other ones for all recursions.

5 Conclusions

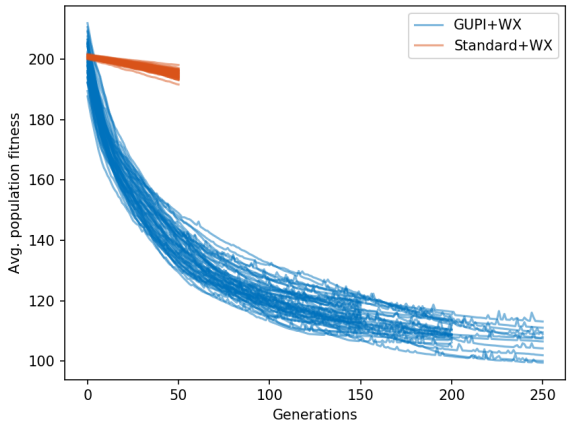
This research work presents an initialization method, GUPI, for grammar-guided genetic programming

Table 3 Descriptive statistics for the G_{RBS} problem regarding the fitness of the final solutions achieved by each initialization+variation GGGP approach (rows) after 50 executions run for each target derivation tree size, represented by columns labeled from 5 to 250 recursions. In bold, the means with significantly lower values for each column and the same variation operator

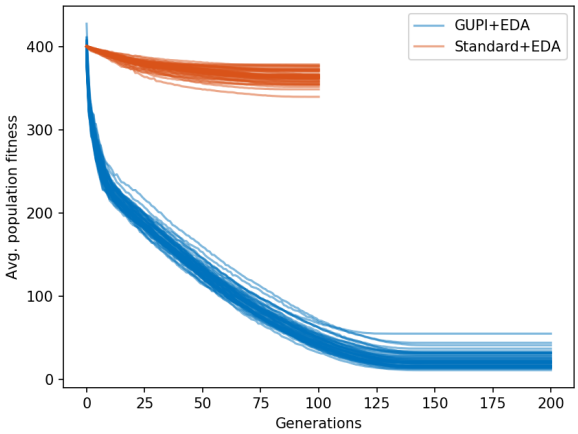
Statistic	Init.+Variation	Recursions					
		5	10	20	50	100	250
Mean	GUPI+WX	12.05	**14.94**	**46.14**	**111.37**	**235.56**	**569.82**
	Standard+WX	**7.32**	21.62	61.53	195.11	398.90	994.71
	GUPI+EDA	4.72	**9.18**	**15.46**	**12.80**	**23.08**	**107.56**
	Standard+EDA	4.32	20.58	53.64	166.54	364.38	963.66
Std. Dev.	GUPI+WX	5.35	4.52	5.31	5.10	8.16	15.74
	Standard+WX	4.69	6.70	9.97	1.23	1.20	1.44
	GUPI+EDA	2.83	1.99	2.10	7.24	8.95	36.86
	Standard+EDA	2.31	4.35	6.52	7.56	7.92	8.24
Min.	GUPI+WX	1.16	4.70	31.68	99.50	219.25	531.54
	Standard+WX	1.00	6.69	35.70	191.64	396.25	990.75
	GUPI+EDA	0.00	5.00	10.00	4.00	11.00	68.00
	Standard+EDA	0.00	12.00	33.00	146.00	340.00	948.00
95% Conf. Int.	GUPI+WX	[10.56, 13.53]	[13.69, 16.20]	[44.67, 47.62]	[110.0, 112.8]	[233.3, 237.8]	[565.5, 574.2]
	Standard+WX	[6.02, 8.62]	[19.76, 23.47]	[58.77, 64.29]	[194.8, 195.5]	[398.6, 399.2]	[994.3, 995.1]
	GUPI+EDA	[3.94, 5.50]	[8.63, 9.73]	[14.88, 16.04]	[10.79, 14.81]	[20.60, 25.56]	[97.34, 117.8]
	Standard+EDA	[3.68, 4.96]	[19.37, 21.79]	[51.83, 55.45]	[164.4, 168.6]	[362.1, 366.6]	[961.4, 965.9]
Max.	GUPI+WX	20.66	22.74	55.64	121.72	256.80	597.67
	Standard+WX	18.48	33.66	79.57	198.10	401.14	997.85
	GUPI+EDA	10.00	13.00	21.00	34.00	55.00	283.00
	Standard+EDA	10.00	28.00	68.00	181.00	379.00	977.00

Table 4 Descriptive statistics for the G_{SR} problem regarding the fitness of the final solutions achieved by each initialization+variation GGGP approach (rows) after 50 executions run for each target derivation tree size, represented by columns labeled from 5 to 250 recursions. In bold, the means with significantly lower values for each column and the same variation operator

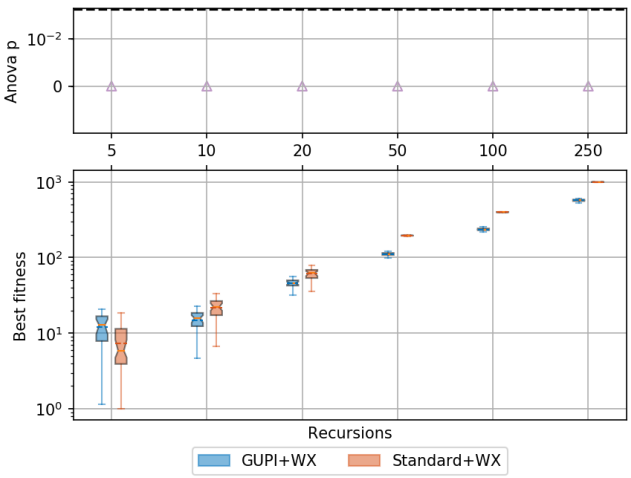
Statistic	Init.+Variation	Recursions					
		5	10	20	50	100	250
Mean	GUPI+WX	5.76	**16.53**	38.38	**121.46**	**265.36**	**696.88**
	Standard+WX	4.75	18.01	39.43	131.90	326.69	853.29
	GUPI+EDA	7.02	**15.84**	**42.58**	**119.34**	**268.82**	**672.34**
	Standard+EDA	7.36	17.32	45.20	126.42	302.26	799.40
Std. Dev.	GUPI+WX	2.77	2.99	3.93	5.03	6.71	13.53
	Standard+WX	2.74	2.50	5.26	9.98	11.78	3.37
	GUPI+EDA	1.57	2.28	3.06	4.78	6.15	13.10
	Standard+EDA	1.85	2.53	3.24	7.27	14.52	17.78
Min.	GUPI+WX	1.00	7.05	28.91	103.28	251.06	657.40
	Standard+WX	1.00	12.74	24.88	109.91	269.48	846.43
	GUPI+EDA	4.00	9.00	36.00	107.00	256.00	628.00
	Standard+EDA	3.00	11.00	39.00	109.00	265.00	751.00
95% Conf. Int.	GUPI+WX	[4.99, 6.53]	[15.70, 17.36]	[37.29, 39.47]	[120.1, 122.9]	[263.5, 267.2]	[693.1, 700.6]
	Standard+WX	[3.99, 5.51]	[17.32, 18.71]	[37.97, 40.88]	[129.1, 134.7]	[323.4, 330.0]	[852.4, 854.2]
	GUPI+EDA	[6.58, 7.46]	[15.21, 16.47]	[41.73, 43.43]	[118.0, 120.7]	[267.1, 270.5]	[668.7, 676.0]
	Standard+EDA	[6.85, 7.87]	[16.62, 18.02]	[44.30, 46.10]	[124.4, 128.4]	[298.2, 306.3]	[794.5, 804.3]
Max.	GUPI+WX	9.73	21.97	45.97	129.96	279.10	719.07
	Standard+WX	9.87	21.98	46.08	160.99	333.91	860.90
	GUPI+EDA	10.00	20.00	50.00	128.00	281.00	696.00
	Standard+EDA	11.00	23.00	55.00	145.00	333.00	831.00



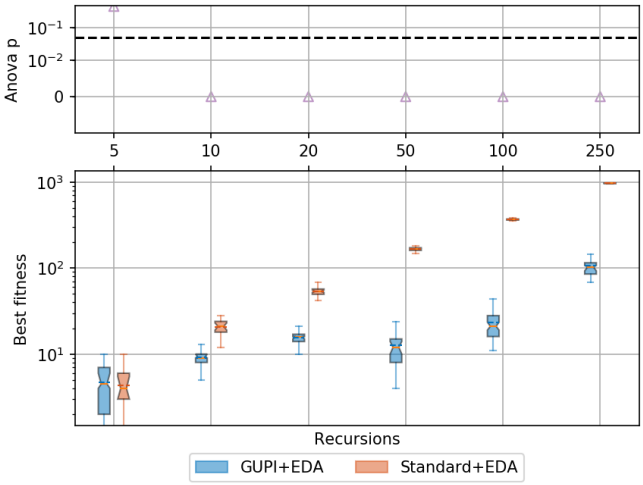
(a) WX. Target derivation trees with 50 recursions.



(b) EDA. Target derivation trees with 100 recursions.



(c) Statistical comparisons using WX as the GGGP variation operator.

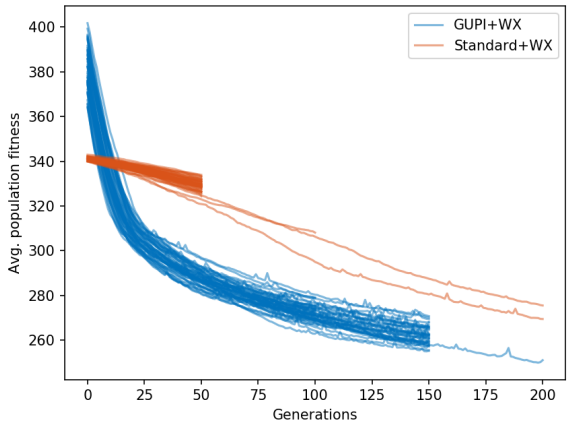


(d) Statistical comparisons using EDA as the GGGP variation operator.

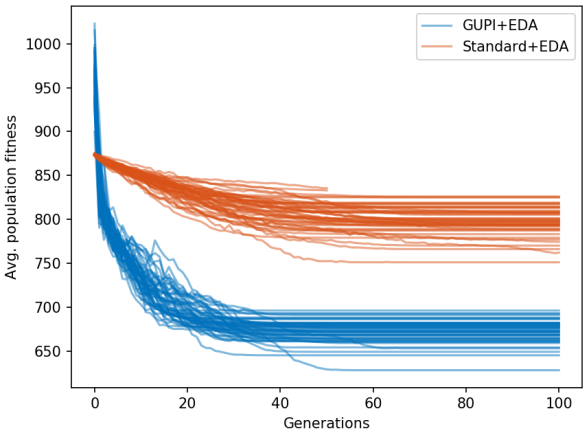
Fig. 5 G_{RBS} problem. 50 executions run for each initialization algorithm, variation operator and target derivation tree size (recursions). Above, the evolution of the average population fitness using either WX or EDA when starting from a population initialized by GUPI (blue) and the standard initialization algorithm (orange). Below, box-and-whisker plots and significance levels p achieved by the t-tests performed to compare GUPI and the standard initialization approach in terms of the fitness of the final solutions achieved by the evolutionary process starting from initial populations generated by GUPI and the standard approach. The dashed line points out the significance-level threshold that rejects the means are equal

(GGGP) that generates individuals uniformly distributed in the search space defined by any context-free grammar. The proposed approach refers to uniformity as the genotypic type, so any derivation tree has the same probability of being generated, independently of its size. The goal is to generate initial populations that are representative of the search space to boost the overall evolutionary process performance. GUPI incorporates a code bloat control mechanism without distorting uniformity to work with recursive grammars by limiting the number of recursive production rules that can be applied to generate an individual, instead of the usually employed tree depth bound setting. Unlike stochastic context-free grammars, GUPI dynamically calculates the probabilities of the production rules of the grammar to pursue uniformity. These probabilities depend on each specific grammar and updated according to the context of the individual under creation.

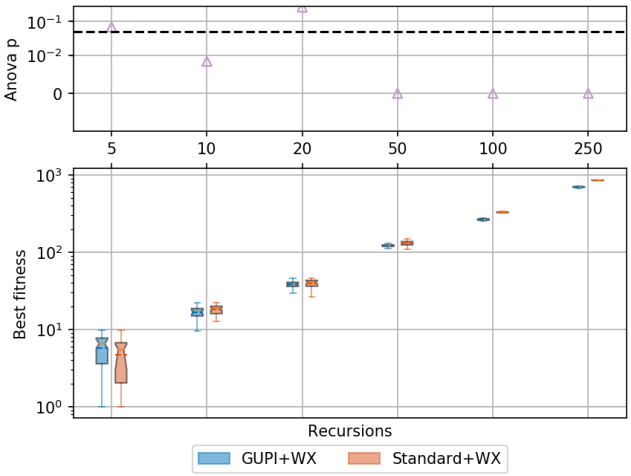
Empirical experiments have been conducted to compare the initial population distribution and the performance of the overall GGGP evolutionary process when starting from initial populations generated by GUPI and a standard frequently used initialization algorithm with code bloat control for GPPP (García Arnau et al.,



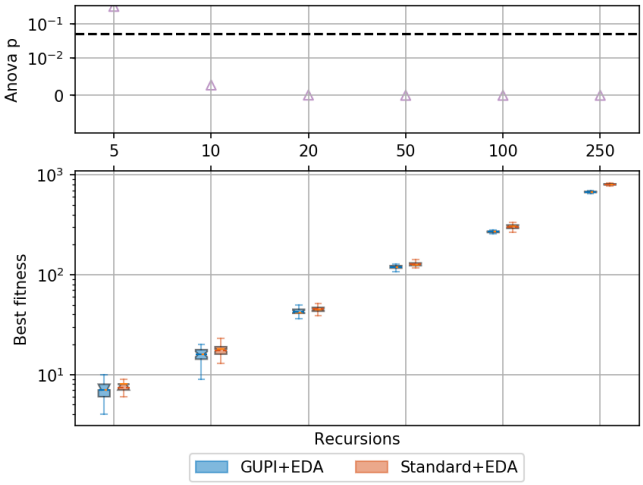
(a) WX. Target derivation trees with 100 recursions.



(b) EDA. Target derivation trees with 250 recursions.



(c) Statistical comparisons using WX as the GGGP variation operator.



(d) Statistical comparisons using EDA as the GGGP variation operator.

Fig. 6 G_{SR} problem. 50 executions run for each initialization algorithm, variation operator and target derivation tree size (recursions). Above, the evolution of the average population fitness using either WX or EDA when starting from a population initialized by GUPI (blue) and the standard initialization algorithm (orange). Below, box-and-whisker plots and significance levels p achieved by the t-tests performed to compare GUPI and the standard initialization approach in terms of the fitness of the final solutions achieved by the evolutionary process starting from initial populations generated by GUPI and the standard approach. The dashed line points out the significance-level threshold that rejects the means are equal

2007). Two very often variation operators in GGGP have been considered: a generational replacement estimation of distribution algorithm (EDA) (Kim et al., 2014), which primarily performs a local search, and the Wigham's crossover (WX) (Whigham, 1995), which tends to explore the search space widely. The aim is to observe whether GUPI-initialized populations can improve the evolutionary process, compared to standard initial populations, using variation operators with very different behaviors.

The results report that the proposed initialization approach generates individuals following a uniform distribution in the search space. Both initial population uniformity and code bloat control drive GUPI to population distributions that improve the overall evolutionary process performance, even if recursive CFGs are employed. Since GUPI generates initial representative populations of the search space, the evolutionary algorithm that takes place afterward reduces its exploratory effort. Therefore, its performance, in terms of convergence speed and quality (fitness) of the final solutions, outperforms the results achieved from initial populations generated by the standard approach. Moreover, the larger the search space is, the more significant the

differences are in terms of performance of the evolutionary process starting from GUPI or the standard approach for both EDA and WX variation operators.

It is also noteworthy that GUPI+EDA always outperforms GUPI+WX in terms of convergence speed for all the experiments carried out from 20 or more recursions. Since EDA has better local search capabilities than WX, it gets the most out of a uniformly distributed initial population to lead the evolutionary process towards the most promising points in the search space. This EDA local search boosting is what precisely guides the evolutionary process to local optima when the standard approach initializes the population. Therefore, GUPI+EDA achieves the best results in terms of convergence speed, i.e., a uniformly distributed initial population with a GGGP evolutionary system that boosts local search. These results open evolutionary algorithms to apply them successfully on harder (larger derivation trees and search spaces) problems than usual.

Following this line of research and considering the fundamental role of genetic variation operators to guide the evolutionary process adequately, the authors are investigating new variation operators that not only take advantage of a uniform initial population but also balance the exploration and exploitation (local search) capabilities of the evolutionary algorithm. We intend to adopt a similar approach like the one presented in this article to calculate the probabilities of the production rules with the goal of estimating a probability distribution of the search space from which the variation operator generates the offspring.

Acknowledgements This research was partially funded by Ministerio de Economía, Industria y Competitividad, Spain, research grant number MTM2014-54053-P, and Artificial Intelligence Lab. at Universidad Politécnica de Madrid.

References

Burke E, Gustafson S, Kendall G (2004) Diversity in genetic programming: An analysis of measures and correlation with fitness. IEEE Trans Evol Comput 8(1):47–62

Chellapilla K (1997) Evolving computer programs without subtree crossover. IEEE Trans Evol Comput 1(3):209–216

Couchet J, Manrique D, Porras L (2007) Grammar-guided neural architecture evolution. In: Bio-inspired Modeling of Cognitive Tasks. Second International Work-Conference on the Interplay Between Natural and Artificial Computation (IWINAC07) Part I, La Manga del Mar Menor, Spain, pp 223–240

Crane EF, McPhee NF (2006) The Effects of Size and Depth Limits on Tree Based Genetic Programming, Springer US, Boston, MA, pp 223–240

Darwin C (1959) On the Origin of the Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life. John Murray, London

Deng W, Zhao H, Zou L, Li G, Yang X, Wu D (2016) A novel collaborative optimization algorithm in solving complex optimization problems. Soft Comput 21(15):4387–4398, DOI 10.1007/s00500-016-2071-8

Deng W, Zhao H, Yang X, Xiong J, Sun M, Li B (2017) Study on an improved adaptive PSO algorithm for solving multi-objective gate assignment. Appl Soft Comput 59:288–302, DOI 10.1016/j.asoc.2017.06.004

Deng W, Xu J, Zhao H (2019) An improved ant colony optimization algorithm based on hybrid strategies for scheduling problem. IEEE Access 7:20281–20292, DOI 10.1109/ACCESS.2019.2897580

Fagan D, Fenton M, O'Neill M (2016) Exploring position independent initialisation in grammatical evolution. In: 2016 IEEE Congress on Evolutionary Computation (CEC), Vancouver, BC, Canada, pp 5060–5067

García Arnau M, Manrique D, Ríos J, Rodríguez Patón A (2007) Initialization method for grammar-guided genetic programming. Knowledge-Based Syst 20(2):127–133

Harper R (2010) Ge, explosive grammars and the lasting legacy of bad initialisation. In: IEEE Congress on Evolutionary Computation, Barcelona, Spain, pp 1–8

Hassanat A, Surya Prasath V, Abbadi M, Abu-Qdari S, Faris H (2018) An improved genetic algorithm with a new initialization mechanism based on regression techniques. Information 9(167), DOI 10.3390/info9070167

Hien N, Hoai N (2006) A brief overview of population diversity measures in genetic programming. In: Pham TL, Le HK, Nguyen XH (eds) Proceedings of the Third Asian-Pacific Workshop on Genetic Programming, Hanoi, Vietnam, pp 128–139

Hopcroft J, Motwani R, JD U (2006) Introduction to Automata Theory, Languages and Computation, 3rd edn. Addison-Wesley Longman Publishing, Boston, MA

Kari L, Rozenberg G (2008) The many facets of natural computing. Commun ACM 51(10):72–83

Kim K, McKay R (2013) Stochastic diversity loss and scalability in estimation of distribution genetic programming. IEEE Trans Evol Comput 17(3):301–320

Kim K, Shan Y, Nguyen X, McKay R (2014) Probabilistic model building in genetic programming: A critical review. Genet Program Evol Mach 15(2):115–167

Koza J (1992) Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA

Koza JR, Keane MA, Streeter MJ, Mydlowec W, Yu J, Lanza G (2006) Genetic Programming IV: Routine Human-Competitive Machine Intelligence. Springer Science & Business Media, New York

Krithivasan K (2009) Introduction to Formal Languages, Automata Theory and Computation. Pearson Education, India

McKay R, Hoai N, Whigham P, Shan Y, O'Neill M (2010) Grammar-based genetic programming: a survey. Genet Program Evol Mach 11(3-4):365–396

Moll RN, Arbib MA, Kfoury AJ (2012) An introduction to formal language theory. Springer Science & Business Media, New York

Murphy E, Hemberg E, Nicolau M, O'Neill M, Brabazon A (2012) Grammar bias and initialisation in grammar based genetic programming. In: Moraglio A, Silva S, Krawiec K, Machado P, Cotta C (eds) Genetic Programming, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 85–96

Nicolau M (2017) Understanding grammatical evolution: initialisation. Genet Program Evol Mach 18:467–507

Nicolau M, Fenton M (2016) Managing repetition in grammar-based genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016, ACM, New York, NY, USA, GECCO '16, pp 765–772

Poli R, Langdon W, McPhee N, Koza J (2008) A Field Guide to Genetic Programming. Lulu.com, UK

Ramos Criado P (2017) New techinques for grammar guided genetic programming: dealing with large derivation trees and high cardinality terminal symbol sets. PhD thesis, Universidad Politécnica de Madrid, Spain

Roth R (2006) Introduction to Coding Theory, Cambridge University Press, Cambridge, UK, p 298

Schweim D, Thorhauer A, Rothlauf F (2018) On the Non-uniform Redundancy of Representations for Grammatical Evolution: The Influence of Grammars, Springer International Publishing, Cham, pp 55–78

Sipser M (2013) Introduction to the Theory of Computation, 3rd edn. Cengage Learning, Boston, MA

Tanev I (2004) Implications of incorporating learning probabilistic context-sensitive grammar in genetic programming on evolvability of adaptive locomotion gaits of snakebot. In: Proceedings of GECCO 2004, Seatle, Washington, pp 155–166

Thorhauer A (2016) On the non-uniform redundancy in grammatical evolution. In: Handl J, Hart E, Lewis PR, López-Ibáñez M, Ochoa G, Paechter B (eds) Parallel Problem Solving from Nature – PPSN XIV, Springer International Publishing, Cham, pp 292–302

Vanneschi L, Castelli M, Silva S (2014) A survey of semantic methods in genetic programming. Genet Program Evol Mach 15(2):195–214

Whigham P (1995) Grammatically-based genetic programming. In: Rosca JP (ed) Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications, Tahoe City, California, USA, pp 33–41