Estimation of Distribution Algorithm for Grammar-Guided Genetic Programming

Pablo Ramos Criado pablo.ramos@aturing.com
Aturing Research, Salamanca, Spain

D. Barrios Rolanía dolores.barrios.rolania@upm.es
Depto. Matemática Aplicada a la Ingeniería Industrial, ETSI Industriales, Universidad Politécnica de Madrid, Spain

David de la Hoz david.delahoz.galiana@alumnos.upm.es
Depto. Inteligencia Artificial, ETSI Informáticos, Universidad Politécnica de Madrid, Spain

Daniel Manrique daniel.manrique@upm.es
Depto. Inteligencia Artificial, ETSI Informáticos, Universidad Politécnica de Madrid, Spain

https://doi.org/10.1162/evco_a_00345

Abstract

Genetic variation operators in grammar-guided genetic programming are fundamental to guide the evolutionary process in search and optimization problems. However, they show some limitations, mainly derived from an unbalanced exploration and local-search trade-off. This paper presents an estimation of distribution algorithm for grammar-guided genetic programming to overcome this difficulty and thus increase the performance of the evolutionary algorithm. Our proposal employs an extended dynamic stochastic context-free grammar to encode and calculate the estimation of the distribution of the search space from some promising individuals in the population. Unlike traditional estimation of distribution algorithms, the proposed approach improves exploratory behavior by smoothing the estimated distribution model. Therefore, this algorithm is referred to as SEDA, smoothed estimation of distribution algorithm. Experiments have been conducted to compare overall performance using a typical genetic programming crossover operator, an incremental estimation of distribution algorithm, and the proposed approach after tuning their hyperparameters. These experiments involve challenging problems to test the local search and exploration features of the three evolutionary systems. The results show that grammar-guided genetic programming with SEDA achieves the most accurate solutions with an intermediate convergence speed.

Keywords

Grammar-guided genetic programming, estimation of distribution algorithms, genetic variation operators, local search, locality, search-space exploration.

1 Introduction

Evolutionary computation (Bäck et al., 1997) is a subfield of natural computing (Kari and Rozenberg, 2008) that borrows ideas from natural evolution (Darwin, 1959) to

Correspondence to daniel.manrique@upm.es; dmanrique@fi.upm.es.

Manuscript received: 13 December 2021; revised: 14 December 2022, 8 June 2023, 24 October 2023, and 14 December 2023; accepted: 8 January 2024.

perform search and optimization processes on populations of individuals representing candidate solutions to a specific problem. The individuals belong to the search space, and the set of all solutions encoded by the individuals is the solution space. An encoding scheme establishes the relationship between these two spaces. Genetic Programming (GP) (Koza, 1992) and Grammar-Guided Genetic Programming (GGGP) (Whigham, 1995) are evolutionary algorithms that belong to this discipline. GP employs programs of variable size to encode possible solutions to a problem. Grammar-Guided Genetic Programming is an extension of GP 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 (derivation trees) must satisfy. Hence, GGGP addresses the closure requirement (or solves the closure problem), which means that all individuals generated during the evolutionary process match the problem restrictions (Vanyi and Zvada, 2003; Poli et al., 2008); that is, trees are derivations of the grammar.

GGGP has become crucial as a method for formalizing constraints in GP. It is a branch of interest in evolutionary algorithms research due to its successful applications in different areas (McKay et al., 2010), resulting, in some cases, in patentable inventions (Koza et al., 2006). GGGP has shown great potential in designing knowledge bases of rules and fuzzy rules for medical purposes (Font et al., 2010), and other types of intelligent systems highly useful today, such as Bayesian networks (Font et al., 2011). Neuroevolution is another important field where GGGP has successfully been applied (Barrios Rolanía et al., 2018), especially with the rise of deep artificial neural networks and deep learning. GGGP applications cover a wide diversity of domains, ranging from rule extraction for medical purposes (Wong and Leung, 2000) and architecture (Hemberg et al., 2008) to circuit design (Tetteh et al., 2022) and ecological modeling (McKay et al., 2006).

Like other evolutionary algorithms, GGGP begins by generating the initial population of derivation trees, usually at random, following a particular distribution (Koza et al., 2006; García-Arnau et al., 2007; Ramos Criado et al., 2020). The evolutionary process then takes place, which comprises three primary operations: selection, variation, and replacement. The selection operation chooses some promising individuals of the population, known as parents. The variation operation generates a set of new individuals with inherited characteristics from the parents, known as offspring. Finally, the replacement operator inserts these new individuals into the population and removes other less adapted ones. Algorithm 1 describes the general operation of GGGP, where i_j is an individual of the population I_g, g is the iteration number (generation), and f_j is its fitness value. It is assumed a minimization problem, where $\forall i_k \in I_g, k \neq j$, if $f(i_j) < f(i_k)$, then i_j is better adapted or has a better fitness than i_k.

The variation operators are crucial to guide the evolutionary process towards the optimum solution or its neighborhood. Therefore, they must perform a local search in promising areas of the search space, focusing on parent individuals' characteristics to produce new ones with similar characteristics. Nevertheless, variation operators must also explore (global search) the search space to generate new diverse individuals, differing from those already existing in the current population. If the variation operators are excessively local, the evolutionary algorithm is likely to converge early to suboptimal solutions, presenting difficulties in escaping from them. If, on the other hand, the variation operators are very explorative generating new individuals, the evolutionary algorithm will take too long to find the optimal solution. Therefore, the performance of the evolutionary process relies to a large extent on its variation operators having

Algorithm 1 Grammar-guided genetic programming. The initialization method, context-free grammar G, population size λ, fitness function $f(i_j) = f_j$, stop criteria, selection, variation, mutation with probability p_m, and replacement operators are the inputs.

```
1:  Generate $\lambda$ individuals for the initial population $I_0$ using the initialization method following
    the production rules of $G$ to ensure feasibility
2:  **for all** $i_j \in I_0$ **do** calculate their fitness value $f(i_j)$
3:  $g \leftarrow 0$
4:  **while** a stop criterion is not met **do**
5:      Select as many individuals from the population as needed by the variation operator
6:      Apply the variation operator to the selected individuals to generate the offspring $O$
7:      **for all** $o_j \in O$ **do** apply the mutation operation with probability $p_m$ to get the set $O'$
8:      **for all** $o'_j \in O'$ **do** calculate $f(o'_j)$
9:      Apply the replacement operator to insert $O'$ in $I_g$ to generate $I_{g+1}$ with the same size $\lambda$
10:     $g \leftarrow g+1$
    **return** the solution encoded by $i_j \in I_g \mid \forall i_k \in I_g, k \neq j, f(i_j) \leq f(i_k)$
```

an adequate balance between exploitation and exploration; namely, an exploitation–exploration trade-off (McKay et al., 2010). They must be able to explore the search space to find promising areas and eventually escape from local optima, but also focus on those promising areas to find the global optimum.

Crossover and mutation are two variation operators employed in GGGP (Whigham, 1995). They do not provide any control over the exploitation–exploration trade-off. A crossover operation may produce utterly different derivation trees given two similar parents. The result of a mutation operation completely varies depending on the mutated derivation tree node. Mutation of nodes close to derivation tree leaves usually produces small variations, while modification of nodes more proximate to the root produces significant changes. However, the latter is less likely than the former given the structure of a tree (there are more nodes close to the leaves than close to the root) and the fact that the choice of the mutation node is random. Additionally, the mutation operation acts with very low probability. Therefore, its influence on the evolutionary process is minimal, and evolution mainly relies on the crossover operator. Whigham's (1995) crossover (WX) has been widely tested and, in most cases, achieves satisfactory results (Couchet et al., 2007). However, there is still margin for improvement (White et al., 2013) since the offspring produced by this operator might not be similar to its parents. Because of this fact, the optimization may focus not on exploiting promising individuals but on exploring new search space areas (Ramos Criado, 2017). Thus, the evolutionary process may show erratic behavior, and some difficulties in progressing towards the optimal solution arise. This common issue in GGGP is related to the genotypic or syntactic locality of crossover operators (Uy et al., 2010), namely, the crossover's ability to perform small changes to the genotype (Galván-López et al., 2009, 2011; Galván et al., 2013).

GGGP has also been criticized for being a very restrictive environment. New variation operators are rarely designed, since they have to deal with derivation trees with a fixed structure and grammar constraints (McKay et al., 2010). Several research lines have focused on addressing these limitations, such as the linearization of CFG derivation trees (O'Neill and Ryan, 2003; Ryan et al., 1998) and the replacement of CFG by tree-adjoining grammars (Joshi and Schabes, 1997). These techniques have been proposed to change the encoding scheme and provide an enhanced environment where new variation operators can be designed. Other approaches are related to the development of improved GGGP optimization methods, as in ant-colony (Dorigo et al., 2006) or grammatical swarm (O'Neill and Brabazon, 2006) algorithms. Another promising optimization technique for GGGP are estimation of distribution algorithms (EDA) (Hauschild and Pelikan, 2011).

1.1 Estimation of Distribution Algorithms

Estimation of distribution algorithms (EDA) use probabilistic models to drive the evolutionary process towards promising solutions. In each iteration, EDA learn a probabilistic model that shapes the distribution of the current population or some selected individual's characteristics. Then, they replace the existing population or a subset with a new one, sampled according to the probabilistic model previously learned. Therefore, EDA base the evolutionary process on learning probabilistic models from selected individuals to guide the generation of new ones.

EDA-GP approaches employ probabilistic models to improve genetic programming algorithms' performance and scale up with the problem size (Sastry and Goldberg, 2006; Shan et al., 2003). Tree encoding is the primary technique to encode the probabilistic models in GP (Kim et al., 2014). Some approaches based on Bayesian principles (Hasegawa and Iba, 2008) learn computer programs that control the distribution of instances generated throughout a tree representation (Looks et al., 2005). This latter work represents programs as binary trees, called zigzag trees, but they possess limited representation capabilities. Probabilistic incremental program evolution (PIPE) (Salustowicz and Schmidhuber, 1997) uses standard GP functions and generates successive populations according to an adaptive probability distribution over the search space. It assumes the independence of tree nodes to reduce the complexity of the probabilistic model and employs pruning methods. However, these assumptions may cause genetic loss, since promising subtrees may not be produced. In most cases, GP tree approaches suffer from the closure problem and may generate infeasible individuals.

Some EDA-GGGP methods apply stochastic CFG to learn the probabilistic model that drives the GGGP evolutionary process (Ratle and Sebag, 2001, 2002; Tanev, 2004). However, the probability of selecting any production rule of a CFG or stochastic CFG at a precise depth of a derivation not only depends on the likelihood of choosing that production rule but also involves the probabilities of selecting the previous production rules to reach the current depth (McKay et al., 2010). Therefore, the probabilistic model may be biased, especially in recursive CFG.

In general terms, the EDA optimization process displays a fast convergence to optimal solutions (Kim and McKay, 2013; Kim et al., 2014). However, there are still limitations related to the exploitation–exploration trade-off that avoid substantial improvements in its performance (O'Neill et al., 2010). Unlike GGGP crossover operators, there is no subtree recombination in EDA-GGGP. Instead, it generates new individuals from the root to the leaves, following the estimated distribution of parent-derivation-trees. Consequently, EDA and even incremental EDA show a reduction of population diversity that may lead to an excessively local search and increase in the probability to converge to local optima (Ramos Criado, 2017).

1.2 Contributions

This paper presents a smoothed estimation of distribution algorithm for grammar-guided genetic programming, SEDA. It has two essential features that distinguish it from other EDA approaches to provide an adequate trade-off between exploration and exploitation (local search). First, SEDA calculates an extended version of the CFG that encodes the solution space. This context-free grammar expansion (CFGE) represents the search space of the problem at hand in a graph-like structure that stores the probabilities of the distribution model, maintaining the dependence of node information. These probabilities are calculated from a subset of selected promising individuals of the current population. This definition simplifies the representation of the search space and facilitates its application or extrapolation to different search techniques. Second, SEDA applies a smoothing method to reduce the *spikes* in the calculated probability distribution by slightly increasing the low probabilities and decreasing the high probabilities accordingly. Then, SEDA generates new individuals with representative characteristics present in the population by following the estimated smoothed probability distribution.

SEDA adopts an EDA approach to perform a local search, since it generates new individuals from a probabilistic model learned from the current population. At the same time, SEDA's smoothing method increases the genetic diversity to avoid premature convergence to suboptimal solutions. Moreover, a hyperparameter is provided to control the smoothing process, and therefore the exploitation–exploration trade-off in the optimization process. These two features, working together, provide crucial support to adequately guide the evolutionary process.

Experiments have been conducted to compare the overall performance of the GGGP evolutionary process when using WX, an incremental EDA, and SEDA. The probabilistic model of the incremental EDA approach is based on the CFGE employed in SEDA to consider the dependence of nodes (dependency-aware). Thus, the incremental EDA involved in the experiments is similar to SEDA but does not employ the smoothing method to show its positive impact. The results show that, although SEDA performs more evaluations than incremental EDA, the former achieves more accurate results. Hence, incremental EDA exhibits a lack of exploration that leads the evolutionary process to fall into local optima, which is prevailed by SEDA, especially when large derivation trees are involved. WX achieves, in most cases, unsatisfactory results in both convergence speed and accuracy of the final solutions when dealing with large search spaces, thus revealing its excessive exploration capabilities.

The rest of the paper is structured as follows: Section 2 defines the cardinality of a production rule and symbol as the base for discussing the concept of locality in genetic variation operators and its relation to the exploitation–exploration trade-off problem. Following this, Section 3 details the proposed smoothed-EDA, along with an example. The experimentation process, in Section 4, is broken down into two stages and involves three different challenging problems. Section 4.1 describes the setup stage, consisting of tuning the hyperparameters of the three GGGP approaches to be compared. Then, Section 4.2 discusses the performance comparison results gathered from six optimization experiments for each of the three problems under study. Finally, Section 5 provides some concluding remarks, contributions, and future lines of research.

2 Locality and the Exploitation–Exploration Trade-Off Problem

This section discusses two techniques commonly used in EA from the point of view of their exploration and local search capabilities: GGGP with WX and EDA. The former tends to overly explore the search space, which might hinder the convergence of the algorithm, while the latter boosts local search, which might lead the evolutionary process to local optima.

Let $G = (V, \Sigma, R, S)$ be a CFG, where V is the nonterminal symbols set, Σ is the terminal symbols set, S is the axiom of the grammar, and R is the set of production rules of the form $A ::= \alpha$ such that $A \in V$ and $\alpha \in (V \cup \Sigma)^*$. The asterisk represents the Kleene closure operation.

The cardinality of a production rule or symbol is the number of different terminal string derivations $A \overset{+}{\Rightarrow} s$, $s \in \Sigma^*$ that can be produced starting from that production rule or symbol. $\overset{+}{\Rightarrow}$ notes the transitive closure of $\Rightarrow$, where one or more production rules are applied. The cardinality of a production rule or symbol x is denoted as $|x|$. The cardinality of a production rule or symbol may not be infinite, even if the derivation involves recursive productions, because GGGP generally sets restrictions on the sizes of the derivation trees to avoid code bloat.

For each derivation $S \overset{+}{\Rightarrow} s$, the cardinality pair $(V_{S \overset{+}{\Rightarrow} s}, m)$ is defined, where $V_{S \overset{+}{\Rightarrow} s}$ is the set of nonterminal symbols in $S \overset{+}{\Rightarrow} s$, and the function $m : V_{S \overset{+}{\Rightarrow} s} \rightarrow \mathbb{N}$ defines the number $m(A)$ of occurrences for each nonterminal symbol $A \in V_{S \overset{+}{\Rightarrow} s}$.

Variation operators are intended to lead the evolutionary process towards new promising individuals so that the new populations are expected to improve the previous populations' overall fitness. However, this requirement is not easy to meet, and, in fact, many variation operators do not achieve it. GGGP crossover operators, as WX, are genetic-based variation operators that usually rely on swapping subtrees of parent derivation trees to produce a new offspring. In the case of WX, the crossover nodes (the roots of swapping subtrees) must contain the same nonterminal symbol to ensure a syntactically feasible offspring (the closure requirement). Given two parent derivations $S \overset{+}{\Rightarrow} s$ and $S \overset{+}{\Rightarrow} s'$, with $s, s' \in \Sigma^*$, $V_{S \overset{+}{\Rightarrow} s} \cap V_{S \overset{+}{\Rightarrow} s'}$ is the set of nonterminal symbols that belong to both parent derivations and, therefore, can be selected as crossover nodes. If the cardinality of the nonterminal symbol within a crossover node is low, then the offspring individuals are likely to be similar to their parents, as small changes to the genotype are usually expected from nonterminals with low cardinality. The property of a variation operator that produces small changes in the genotype of the offspring (with respect to the parents) is known as genotypic locality. An exploitation behavior appears when the locality is maintained, and an exploration behavior when it is not. When the cardinality of the crossover node is high, the locality is less likely to be maintained. According to this reasoning, genotypic locality is unlikely to be maintained for large search spaces, which are actually the most useful in real-world applications.

If $S \overset{+}{\Rightarrow} s$ and $S \overset{+}{\Rightarrow} s'$ are two parent derivations with their corresponding cardinality pairs $(V_{S \overset{+}{\Rightarrow} s}, m)$ and $(V_{S \overset{+}{\Rightarrow} s'}, m')$, the number of different crossover operations that can be performed using WX, $X_{(S \overset{+}{\Rightarrow} s, S \overset{+}{\Rightarrow} s')}$, is calculated as

$$|X_{(S \overset{+}{\Rightarrow} s, S \overset{+}{\Rightarrow} s')}| = \sum_{A \in V_{S \overset{+}{\Rightarrow} s} \cap V_{S \overset{+}{\Rightarrow} s'}} m(A) m'(A). \tag{1}$$

The number of different crossover operations increases according to the number of different feasible crossover points in parent derivations, represented by the cardinal of the set $V_{S \overset{+}{\Rightarrow} s} \cap V_{S \overset{+}{\Rightarrow} s'}$. The cardinality of the offspring derivations set that WX can produce from $S \overset{+}{\Rightarrow} s$ and $S \overset{+}{\Rightarrow} s'$ is up to $|X_{(S \overset{+}{\Rightarrow} s, S \overset{+}{\Rightarrow} s')}|$.

Let us consider the recursive CFG G_{rec} as an example for clarification purposes, defined as

$$G_{rec} = (V, \Sigma, R, S)$$
$$V = \{S, Recursive\}$$

$$\begin{aligned}
&\Sigma = \{1\} \\
&R = \{ \\
&\qquad r_1 : \quad S ::= Recursive \\
&\qquad r_2 : \quad Recursive ::= 1\ Recursive \\
&\qquad r_3 : \quad Recursive ::= 1 \\
&\}. \qquad\qquad\qquad\qquad\qquad\qquad (2)
\end{aligned}$$

Since only one nonterminal symbol, $Recursive$, can be selected as crossover point in G_{rec} derivations, then from (1),

$$|X_{(S \overset{+}{\Rightarrow} 1^n, S \overset{+}{\Rightarrow} 1^{n'})}| = m(Recursive)m'(Recursive).$$

Note that, even for this simple example, when parents show 10 possible crossover points each, the number of different crossover operations is already 100. The cardinality of the offspring derivations set for this particular case is 20, with the largest derivation having 20 recursions and the smallest having 1.

2.1 Estimation of Distribution Algorithms Behavior

The performance of the EDA optimization process mainly relies on the probabilistic model abstraction. A more detailed representation of the population distribution, which stores more information about the derivation tree structure, typically the location or dependence of the nodes (McKay et al., 2010), tends to produce individuals that are likely to be similar to previous promising individuals. As a result, it boosts the exploitative (local search) behavior. Therefore, it is more probable that it will converge to local optima since close search space areas are more likely to be explored. On the contrary, a more vague representation of the population distribution, which stores less information about the derivation tree structure—for example, assuming the independence between nodes (Salustowicz and Schmidhuber, 1997)—facilitates the production of extremely different individuals from their ancestors. Subsequently, the optimization process shows a more exploratory behavior that reduces the probability of converging to local optima but also the convergence speed. In most cases, the EDA approaches for GP provide a more detailed representation of the population distribution that reduces the optimization process exploratory behavior. According to this, applying additional methods, such as mutation, that increase the exploratory behavior to search for new individuals plays an essential role in EDA approaches for GP.

EDAs sample new individuals following a probabilistic model. In the case of adopting a generational replacement strategy, which is opposite to the WX approach, EDAs produce a new whole population that replaces the previous generation. As a result, only the characteristics that are the most widespread in the population are likely to be transferred to the next generation. Consequently, the probabilistic model iteratively focuses on these characteristics. Furthermore, using a detailed representation of the population distribution can produce substantial diversity loss, which means that EDA may potentially lose promising individuals that will probably not be reproduced in the following generations.

Table 1 shows comparisons of the average number of evaluations performed to reach a stop criterion, using GGGP with an incremental EDA approach and WX as variation operators, when searching for specific uniformly generated target derivation trees of G_{rec} with 5 and 10 recursions. The stop criterion is met for both GGGP approaches

Table 1: Average number of evaluations to meet a stop criterion and average fitness (3) of the solutions achieved by GGGP with G_{rec} (2) using WX and incremental EDA. 100 executions are run for each of the two evolutionary approaches and target derivation tree size. Standard deviations in brackets.

	5 recursions		10 recursions	
	Evaluations	Fitness	Evaluations	Fitness
EDA	3,200 (4,664.8)	3.89 (0.35)	9,800 (1,400)	5.79 (1.21)
WX	9.72 (20.23)	0.32 (0.47)	136.24 (58.89)	3.76 (1.23)

when the target derivation tree is found or 10,000 evaluations have passed. WX selects two individuals from the population and produces two new ones due to the parents' subtree swapping. Incremental EDA chooses 50% of the population to generate the same number of individuals as offspring. Since WX generates fewer individuals than EDA in each generation, it is unfair to use generations as the time unit. Instead, the number of evaluations that the fitness function performs in each generation is employed, which matches the number of new individuals produced.

Table 1 also compares the average fitness (3) of the solutions achieved. Both GGGP approaches search for the same target derivation trees. Focusing on the example under study G_{rec}, a derivation tree has n recursions when the production r_2 has been rewritten n times. The GGGP fitness function $f(s, s')$ calculates the Levenshtein distance between the word $s = a_1a_2 \ldots a_n$ encoded in the target derivation tree and the word $s' = a'_1, a'_2 \ldots a'_m$ encoded by the individual to evaluate the minimum number of single-character edits required to change one word into the other,

$$f(s, s') = \sum_{i=1}^{max\{n,m\}} d(a_i, a'_i), \tag{3}$$

where

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

Therefore, $f(s, s')$ is the individual fitness that encodes s' and the evolutionary process seeks to minimize the fitness with an optimum value of 0.

The population size is 100 individuals, initially generated following the production rules until reaching a terminal string derivation. When there are several production rules with the same nonterminal symbol on their left-hand side (r_2 and r_3 in G_{rec}), selecting any is equiprobable. Both GGGP approaches employ the tournament selection of size 5 to select the best-fit individual out of the five. EDA performs a number of tournaments equivalent to 50% of the population size to calculate the probabilistic model, while WX executes the two necessary to carry out the crossover. They do not use mutation to reduce the evolutionary-process random component and facilitate the comparison. Replacement substitutes the worst individuals with the offspring, keeping the population size constant. 100 executions have been performed for each experiment. Table 2 summarizes the hyperparameters employed.

Table 1 reports that WX consistently outperforms EDA, since WX achieves lower fitness values than EDA with fewer evaluations. WX can recombine parent derivation

Table 2: Hyperparameters employed in the comparisons shown in Table 1.

Pop.	Initialization	Selection	Replacement	Execs.	Stop criteria
100	Equiprobable	Tournament (5)	Sub. the worst indiv. with off.	100	Target found or 10,000 evaluations

trees to produce new derivation trees with a different number of recursions that may improve the individuals' fitness. The EDA probabilistic model likely generates small trees because the population initialization is biased to derivation trees with few recursions. Note that the likelihood of generating a derivation tree with five or more recursions in the initial population is 0.5^5. Therefore, EDA is unable to produce larger derivation trees even if more evaluations are performed.

In addition, EDA never finds the target derivation trees when increasing their size to 20 or more recursions, always reaching the 10,000-evaluations stop criterion. This result suggests that EDA gets trapped in local optima, shallow derivation trees with fitnesses greater than zero (the optimum) surrounded by others similar in number of recursions with worse fitness, unable to escape because of its excessive local search capability. On the contrary, although WX takes more evaluations to stop when dealing with large target derivation trees, it can find the solution. However, WX achieves solutions with an exponential growth in average fitness and evaluations as the size of the target derivation tree increases, which indicates that excessive exploration cannot deal with large search spaces either.

3 The Smoothed Estimation of Distribution Algorithm

As an estimation of distribution algorithm, SEDA estimates the population distribution to learn or adapt a probabilistic model that encodes the most promising individuals' characteristics. After doing so, SEDA follows the probabilistic model to produce a set of individuals as the new offspring. SEDA also utilizes a smoothing method that aims to create an adequate balance between exploration and exploitation. It increases the offspring diversity to reduce the likelihood of premature convergence. Simultaneously, SEDA's probabilistic-model-based production of individuals enhances the local search capabilities to speed up the evolutionary process.

First, an overview of how SEDA works is provided. Then, the following subsections present a more accurate description. Before the evolutionary process begins, SEDA constructs a tree graph from the specific CFG that defines the search space problem: all derivation trees that the CFG can produce. This graph is the context-free grammar expansion (CFGE). Then, in each generation, SEDA provides the offspring from a set of parent derivation trees selected by the selection operator, which serve as a sample of the current population. This process comprises two main steps. First, SEDA annotates the CFGE according to the parent derivation trees. A probability is assigned to each production rule depending on its location in the CFGE. Thus, the annotated CFGE yields a probabilistic model from which SEDA produces the offspring. These probabilities are calculated based on the absolute frequencies of the applied production rules in each specific node to generate the parent individuals. Then, SEDA applies the smoothing method to the previously calculated frequencies. This method reduces the *spikes* in the probability distribution and avoids null probabilities of non-sampled production rules.

The final probability value of a production rule at a given location depends on the calculated frequency for that production concerning the frequencies of other production rules that could have been applied to the same node. The second step of SEDA comprises the generation of the offspring. The annotated CFGE is traversed from the root to the leaf nodes according to the probabilities previously calculated to generate each individual. The resulting path represents a new derivation tree of the offspring.

3.1 The Context-Free Grammar Expansion

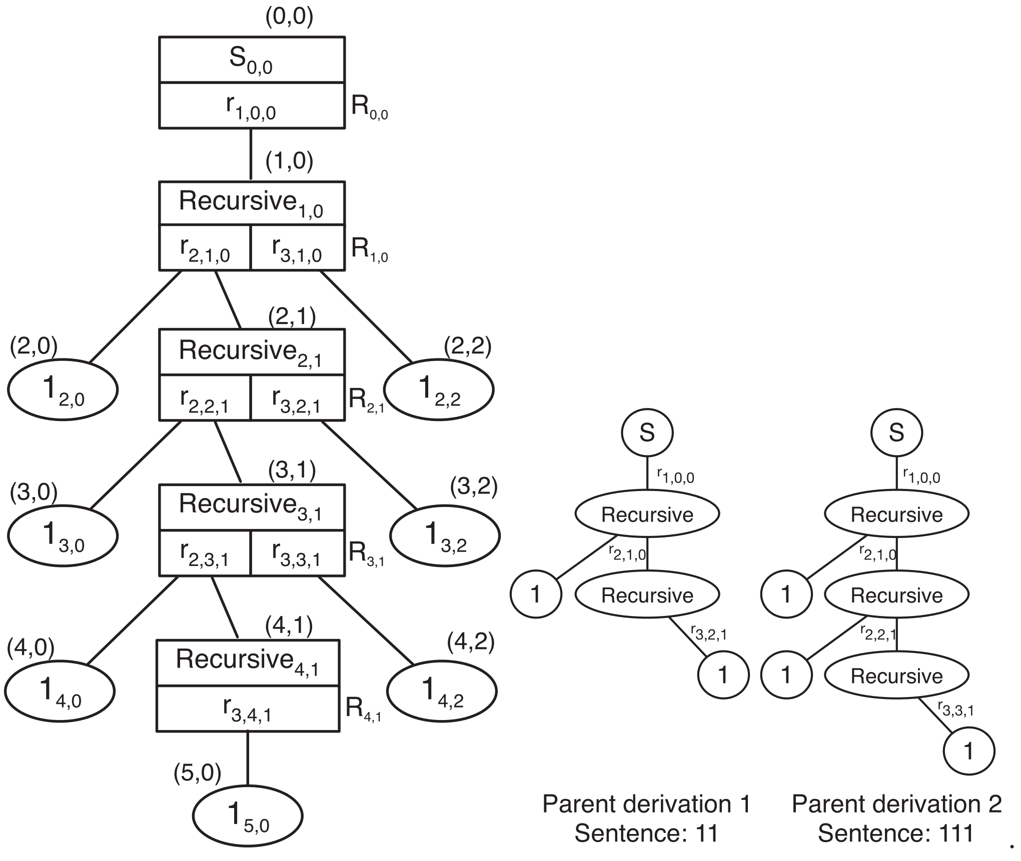
The CFG $G = (V, \Sigma, R, S)$ employed in a given GGGP algorithm is extended to represent a probabilistic model that approximates the current population distribution. The CFG search space, together with the production rules applied to generate each sentence of the grammar language, is represented as a tree graph. This graph-like representation of all derivation trees produced by a CFG is the CFG expansion (CFGE), which also encodes the probabilistic model employed to generate SEDA offspring. Each node of this tree is labeled by the coordinates (d, n), where d is the depth of the node, starting from the root at $d = 0$ to the maximum depth (the number of nodes in the path from the root to the deepest leaf, without taking into account the root), and n is the node number at depth d, numbered from left, $n = 0$, to right. The maximum depth of a CFGE is limited by a recursion bound, which establishes the maximum number of recursive productions (recursions) on each possible derivation tree encoded by the CFGE. Once it is reached, there are a finite number of non-recursive productions needed to reach the leaves. Therefore, although a recursive CFG can generate an infinite-sized CFGE, the recursion bound prevents it from doing so.

A CFGE node of coordinates (d, n) contains either a terminal symbol $a \in \Sigma$, noted as $a_{d,n}$, or a nonterminal $A \in V$, noted as $A_{d,n}$. In the case of the latter, the node also includes the set $R_{d,n} \subset R$ of production rules. As in the case of terminal symbols, each rule $r_i \in R_{d,n}$ is noted by $r_{i,d,n}$ to indicate its integration in the node. This last type of node is called meta-node. The root of the CFGE is a meta-node containing the axiom S of the CFG, $S_{0,0}$, and the set of production rules $R_{0,0}$. The leaves are always nodes with a terminal symbol a located at (d, n), $a_{d,n}$. The intermediate nodes are always meta-nodes similar to the root but are those which contain any nonterminal $A \in V$ located at (d, n), $A_{d,n}$ together with its set $R_{d,n}$. For every $r_{i,d,n} \in R_{d,n}$ in a meta-node, each symbol in α generates a child node or meta-node. If the terminal symbol $a \in \alpha$ is at position n, then the child node $(d + 1, n)$ contains this terminal symbol $a_{d+1,n}$. If the nonterminal $A \in \alpha$ is at position n and the recursion bound is not reached, then the child meta-node $(d + 1, n)$ contains this nonterminal $A_{d+1,n}$ and the set $R_{d+1,n}$. If the recursion bound is achieved for some production rules, then they are removed from $R_{d+1,n}$.

Figure 1a shows the CFGE for G_{rec}, defined in (2), for a recursion bound of 3. Since G_{rec} is recursive, the tree graph might be infinite, but the recursion bound prevents it. Moreover, all recursive productions (r_2 in this example) may be applied up to three times to generate a derivation tree from this CFGE. This means that SEDA implicitly implements a code-bloat control mechanism since derivation trees larger than the CFGE size cannot be generated.

3.2 The Annotated Context-Free Grammar Expansion

Given any CFGE as the example of Figure 1a, SEDA calculates and annotates a probability to each production rule $r_{i,d,n}$ within the meta-nodes. These probabilities together with their location at the CFGE represent the distribution of the current population. The annotated probability of a production rule $r_{i,d,n}$ is the probability of applying r_i located



(a) The CFGE of G_{rec} for a recursion bound of 3 (b) Two parent derivation trees from G_{rec} and the production rules applied in each node

Figure 1: (a) The CFG expansion that represents the G_{rec} search space together with its production rules for a recursion bound of 3. Note that the production rule r_2, the only recursive production in G_{rec}, is rewritten three times. (b) Two parent derivation trees selected to sample the current population and the production rules applied in each node: $r_{i,d,n}$ represents the production rule $r_i \in R$ of G_{rec} applied on the node located at depth d and node number n.

at the meta-node (d, n) to generate a specific derivation for the offspring within all possible derivations starting from the specific nonterminal A in the same meta-node, $A_{d,n}$. The construction of the annotated CFGE comprises three major stages: the calculation of the frequencies, the smoothing of the obtained frequencies, and the calculation of the probabilities of the model. This process begins after the selection operator chooses the subset of parent individuals (derivation trees) as representative of the current population to build the probabilistic model:

1. *The calculation of the frequencies* involves traversing each selected parent individual, from the root (located at depth 0) to the leaves. This process checks what production rule has been applied in each level of depth to each nonterminal node. Each time a production rule $r_i : A ::= \alpha$ is involved in a parent individual,

where A is located at depth d, and node number n, an associated frequency $\varphi_{r_{i,d,n}}$ to the corresponding rule in the CFGE (initially set up to 0) is increased by 1. Thus, once every selected parent derivation tree has been traversed, the CFGE contains the absolute frequencies of the applied rules according to the depth and node number where they were used. In the case of the example of Figure 1a, and considering the two derivation trees shown in Figure 1b to sample the current population, supposedly chosen by the selection operator, the values of these frequencies, taking the meta-node (1,0) as an example, are the following:

$\varphi_{r_{2,1,0}} = 2$ is the frequency for the recursive production rule r_2 of G_{rec}, located at the meta-node (1,0) of the CFGE, $r_{2,1,0}$. This meta-node represents the nonterminal $Recursive$, $Recursive_{1,0}$. The value is 2 because r_2 has been applied at meta-node (1, 0) in both derivation trees to rewrite string $1Recursive$ from $Recursive$.

$\varphi_{r_{3,1,0}} = 0$ is the frequency for the production rule r_3 of G_{rec}, located also at the meta-node (1,0) of the CFGE, $r_{3,1,0}$. This value is 0, as r_3 has never been applied to produce either parent derivation tree at meta-node (1, 0).

This example shows that the frequency value of $r_{3,1,0}$ is zero, $\varphi_{r_{3,1,0}} = 0$. Calculating the probabilistic model from these absolute frequencies results in the generation of new individuals without any chance of applying the production rule r_3 at (1,0). This scenario causes a loss of exploration of the search space, increasing the probability that the evolutionary algorithm achieves a suboptimal solution.

2. A *smoothing* rate $\sigma \geq 0$ in (4) is applied to the previously calculated frequencies $\varphi_{r_{i,d,n}}$ to obtain a smoothed frequency, or weight, associated to each rule $r_{i,d,n}$ of the CFGE. As pointed out in Sections 3.3 and 4, fair values for σ are within the range [0,0.1],

$$\omega_{r_{i,d,n}} = \varphi_{r_{i,d,n}} + \sigma \sum_{r_{j,d,n} \in R_{d,n}} \varphi_{r_{j,d,n}}. \tag{4}$$

Following the example of Figure 1, the values of the weights $\omega_{r_{i,d,n}}$ for the meta-node (1,0) are calculated as follows, using (4) with $\sigma = 0.1$:

$\omega_{r_{2,1,0}} = \varphi_{r_{2,1,0}} + \sigma\varphi_{r_{2,1,0}} + \sigma\varphi_{r_{3,1,0}} = 2.2$ since $R_{1,0} = \{r_{2,1,0}, r_{3,1,0}\}$. Similarly, $\omega_{r_{3,1,0}} = 0.2$.

Note that the smoothed frequency $\omega_{r_{3,1,0}}$ is now 0.2, instead of the absolute frequency $\varphi_{r_{3,1,0}} = 0$, calculated in the previous step. In most cases, if the smoothing rate $\sigma > 0$, then $\omega_{r_{3,1,0}} > \varphi_{r_{3,1,0}}$. Calculating the probabilistic model from the smoothed frequencies provides a non-zero probability to rule r_3 at (1,0), which permits SEDA to apply it when generating the offspring although it has not been involved in the generation of the parent individuals.

3. The last stage of the construction of the annotated CFGE comprises *the calculation of the probabilities*, $\rho_{r_{i,d,n}}$, associated to each production rule $r_{i,d,n}$ of the CFGE. This annotated CFGE conforms the base of a probabilistic model from which the offspring might be obtained by SEDA,

$$\rho_{r_{i,d,n}} = \frac{\omega_{r_{i,d,n}}}{\sum_{r_{j,d,n} \in R_{d,n}} \omega_{r_{j,d,n}}}. \tag{5}$$

If the denominator, $\sum_{r_{j,d,n} \in R_{d,n}} \omega_{r_{j,d,n}} = 0$, then an option is to assign the same probability to all $r_{i,d,n} \in R_{d,n}$, so that

$$\rho_{r_{i,d,n}} = \frac{1}{k}, \tag{6}$$

where k is the cardinality of $R_{d,n}$. It is also possible, under the same condition, to apply any other approach to generate the rest of the derivation tree, like the Grammatically Uniform Population Initialization, which uniformly generates derivation trees (Ramos Criado et al., 2020).

The values of the probabilities $\rho_{r_{i,d,n}}$ for the ongoing example in the meta-node (1,0) are $\rho_{r_{2,1,0}} = \frac{\omega_{r_{2,1,0}}}{\omega_{r_{2,1,0}} + \omega_{r_{3,1,0}}} = 0.92$ while $\rho_{r_{3,1,0}} = 0.08$. This means that, instead of an equiprobability scenario, the recursion rule r_2 will be likely applied at depth 1 to generate the offspring.

Another interesting result is that the production rule r_3 at (1,0) has a probability of 0.08 of being chosen to generate the offspring, despite not intervening in the generation of the parent individuals. If $\sigma = 0$, then $\rho_{r_{2,1,0}} = 1$ and $\rho_{r_{3,1,0}} = 0$. Conversely, the higher σ, the lower $\rho_{r_{2,1,0}}$ and the higher $\rho_{r_{3,1,0}}$, tending both to the value 0.5 in the limit. Therefore, the smoothing factor controls the variety of the offspring regarding their parents, helping balance the exploitation–exploration trade-off.

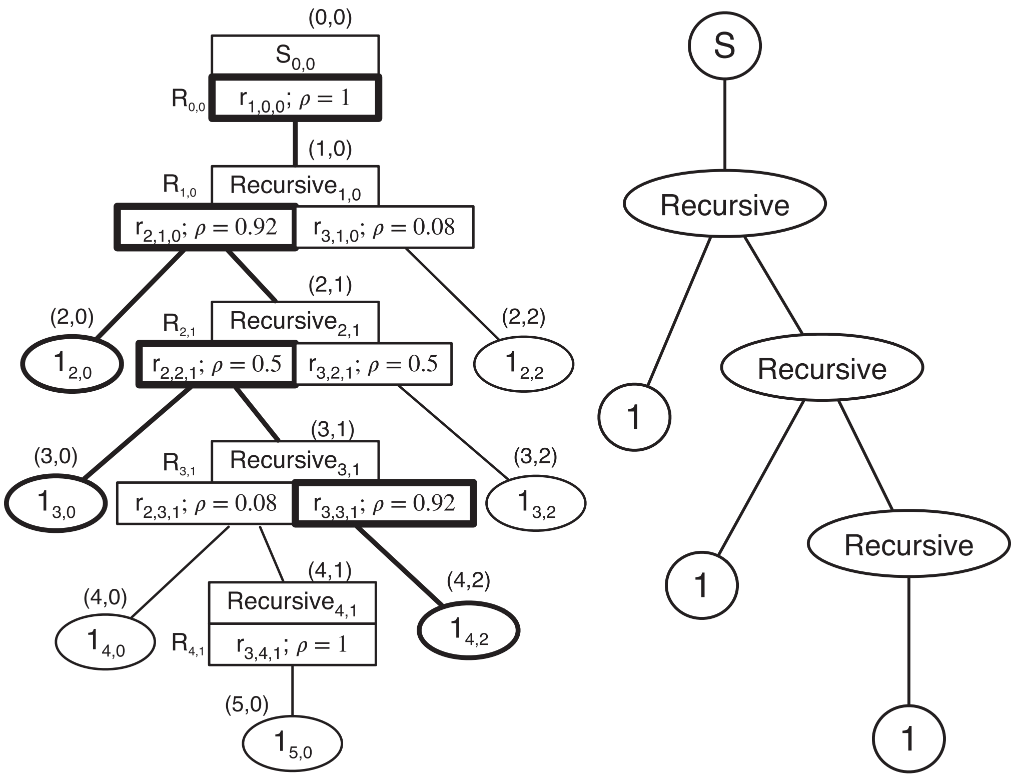
Given the set of parent derivation trees, SEDA might calculate the entire offspring at this point from the probabilistic model represented by the annotated CFGE. Instead, an incremental approach is adopted so that the probabilistic model is updated according to a learning rate of $\mu \in (0, 1)$ to preserve diversity. Considering that SEDA executes once per generation, the probability of applying a production rule $r_{i,d,n}$ in the generation g, $\rho_{g,r_{i,d,n}}$, depends on the probability of using the same production rule in the previous generation $\rho_{g-1,r_{i,d,n}}$, the probabilistic model $\rho_{r_{i,d,n}}$, and the learning rate μ. This is,

$$\rho_{g,r_{i,d,n}} = (1-\mu)\rho_{g-1,r_{i,d,n}} + \mu\rho_{r_{i,d,n}}. \tag{7}$$

At the start of the evolutionary algorithm, when $g = 0$, equiprobability values are assigned to $\rho_{g=0,r_{i,d,n}}$, $\forall r_{i,d,n} \in R_{d,n}$ for each meta-node (d, n).

3.3 The Offspring

According to the probabilistic model, the annotated CFGE is traversed from the root meta-node to the leaf nodes to generate each offspring individual. For every meta-node (d, n) visited, a production rule $r_{i,d,n}$ is applied according to its associated incremental probability $\rho_{g,r_{i,d,n}}$. The resulting traversed path in the annotated CFGE represents a new derivation tree of the offspring. This process is executed as many times as the number of derivation trees are to be generated for the offspring. Figure 2 shows an example of a derivation tree generation using the G_{rec} annotated CFGE after its probabilistic model is updated according to the two-parent derivation trees selected to sample the current population. Figure 2a shows the updated annotated CFGE. The probability $\rho_{g,r_{i,d,n}}$ associated with each production rule $r_{i,d,n}$ of the meta-nodes (d, n) has been updated according to the parent derivation trees. The resulting annotated CFGE is traversed a single time to generate the new derivation tree of Figure 2b. Firstly, in this example, the production rule r_1 in G_{rec}, located at the meta-node (0,0) of the CFGE, $r_{1,0,0}$, is chosen with a probability of 1 (according to the probability model, $\rho_{1,0,0}$) and applied. Then, r_2 at



(a) An example of traversed path on the annotated CFG expansion of G_{rec} (b) The derivation tree obtained from traversing the annotated CFG expansion

Figure 2: (a) An example of traversed path (bold lines) on the resulting annotated CFG expansion after updating the probabilistic model with the selected parent derivation trees of the example. (b) The resulting derivation tree from traversing the annotated CFG expansion of G_{rec}.

meta-node (1, 0), r_2 at meta-node (2, 1), and r_3 at meta-node (3, 1) are consecutively selected and applied. The resulting traversed path is highlighted with bold lines on the annotated CFGE. This path represents the derivation tree of Figure 2b.

Algorithm 2 describes SEDA at generation g to produce an offspring of any number of individuals. The CFGE, smoothing rate σ, and learning rate μ are the inputs, together with a set of parent derivations previously chosen by the selection operator to sample the current population. The algorithm returns as the offspring a new set of derivation trees based on parent characteristics.

The smoothing rate σ in stage 2 of the construction process of the annotated CFGE reduces the probability spikes in the estimated population distribution performed in the third stage. This rate tunes SEDA's behavior regarding its ability to explore the search space to avoid local optima and the local search of similar individuals to speed up the convergence process. $\sigma \in [0, 0.1]$, the smaller the value, the deeper the local search; on the contrary, SEDA's exploration ability increases as σ increases. In the experiments carried out, if $\sigma > 0.1$, SEDA's exploration capability is so high that the algorithm might not converge. The following scenario reveals the essential impact of SEDA's smoothing factor.

Algorithm 2 SEDA at generation g. The CFGE, smoothing rate σ, learning rate μ, a set of parent derivations, and the offspring size are the inputs.

```
1: for all $r_{i,d,n} \in$ CFGE do $\varphi_{r_{i,d,n}} = 0$
2: for all nonterminal $A$ of the parent individuals do        ▷ Calculation of frequencies
3:     The depth $d$ and node number $n$ of $A$ correspond to the meta-node $(d,n) \in$ CFGE
4:     The production rule $r_i : A ::= \alpha$ in the parent individual corresponds to $r_{i,d,n} \in$ CFGE
5:     $\varphi_{r_{i,d,n}} = \varphi_{r_{i,d,n}} + 1$
6: for all meta-node $(d,n) \in$ CFGE do
7:     for all production rule $r_{i,d,n} \in (d,n)$ do
8:         Calculate its smoothed frequency $\omega_{r_{i,d,n}}$, with rate $\sigma$, applying (4)
9:         if $\sum_{r_{j,d,n} \in R_{d,n}} \omega_{r_{j,d,n}} > 0$ then
10:            Calculate the probability $\rho_{r_{i,d,n}}$ applying (5)
11:        else
12:            Apply (6)
13:        Update the incremental probability $\rho_{g,r_{i,d,n}}$, from (7), with the learning rate $\mu$
14: for each individual of the offspring to generate do
15:     Traverse the CFGE from the root, following the incremental probabilities $\rho_{g,r_{i,d,n}}$
16:     for each node or meta-node $(d,n)$ traversed do
17:         Build the terminal $a_{d,n}$ or nonterminal $A_{d,n}$ node for the new derivation tree
18:         Connect the new node to its parent, except for the axiom
19:     Discard the new individual if it is equal to any other already generated for this offspring
```

Suppose a production rule r_i located at the meta-node (d, n) of the CFGE that has not been used, or with a low frequency compared to the rest of productions r_j, $i \neq j$, of the same meta-node, to produce the parent derivation trees. In this case, the frequency value of $r_{i,d,n}$ in the annotated CFGE, $\varphi_{r_{i,d,n}}$, is zero or very low compared to the rest of the frequency values $\varphi_{r_{j,d,n}}$. Suppose also that some or all of the other production rules of the same meta-node, $r_{j,d,n}$, are involved in the generation of the parent derivation trees and, therefore, $\varphi_{r_{j,d,n}} > 0$ for some or all j. According to (5), applying the production rule r_i at (d, n) to generate the offspring is unlikely in this scenario, $\rho_{r_{i,d,n}} = 0$. Under these conditions, the offspring are strictly based on the parents' characteristics, which means that the algorithm performs a local search and tends to prematurely converge to a local optimum. This scenario is usual in EDA, which is equivalent to SEDA with $\sigma = 0$.

4 Experimental Results

The experimentation conducted is comprised of two stages to compare the performance of the overall GGGP evolutionary process when using WX, an incremental EDA, and SEDA in the sixth line of Algorithm 1 (Section 1). WX swaps subtrees of two parent derivation trees whose roots must contain the same nonterminal symbol to generate the new offspring of size two. Given the definition of SEDA in Algorithm 2, EDA can be considered a particular case, where the smoothing rate $\sigma = 0$. Finally, Algorithm 2 describes SEDA.

It is a standard practice in EDA to implement upper and lower thresholds on probabilities to prevent 0 and 1 values in the offspring generation. In contrast, SEDA uniformly transforms all probabilities for each meta-node according to the population distribution regardless of whether the bounds are exceeded. Although bounds in EDA may be a valid alternative, the experiments conducted do not apply such bounds, as their intermittent application might infuse noise into the evolutionary process.

Similar to the comparison carried out in Subsection 2.1, the common GGGP hyperparameters employed with the three approaches are the following: population size of 100 individuals, size 5 tournament selection, replacement of worst individuals with offspring, and no mutation.

The first stage consists of tuning the hyperparameters of each evolutionary algorithm. In the case of GGGP with WX and EDA-GGGP, this stage studies whether the replacement rate influences the quality of the solutions achieved. The replacement rate denotes the proportion of the population to be replaced every generation, that is, the size of the offspring to be generated by WX, EDA, or SEDA. In the case of WX, which generates offspring of size 2, the selection method (line 5 of Algorithm 1) and crossover (line 6 of Algorithm 1) must be repeated as necessary. A generational replacement strategy in WX, which implies high replacement rates, boosts the exploration ability of the evolutionary algorithm. On the contrary, the same replacement approach drives EDA to a diversity loss since most individuals in the next generation follow the same probability distribution. In the case of SEDA, besides the replacement rate, the study also involves the smoothing rate σ since it is an essential hyperparameter to control the balance between exploration and local search.

The second stage employs the best hyperparameters configuration achieved in the previous phase for each GGGP approach to compare their performance in searching for specific derivation trees of different depths with several search space sizes. Performance is measured in terms of fitness of the final solutions and number of evaluations needed to reach the stop criterion. Evaluations are employed instead of generations, since the computational cost of WX generations is lower than EDA-GGGP and SEDA generations. Nevertheless, evaluations of the new generated individuals are comparable in terms of computational cost for the three approaches.

Both stages employ three different benchmark problems to tune the hyperparameters of each evolutionary algorithm and compare their performance, respectively. Varied and difficult real-world problems are selected to provide a set of representative experiments. Table 3 shows the CFG that define these problems with three levels of complexity and different features to observe how the proposed algorithm, SEDA, behaves in common scenarios of evolutionary algorithms. G_{DFFNN} encodes dense, deep feedforward neural network architectures. It contains two recursive production rules: one to determine the number of hidden layers (L), and another to define the number of neurons (N) per layer. The training dataset determines the number of input and output neurons. The terminal / (slash) separates layers and n represents a neuron. The second problem employs G_{RBS} to encode knowledge bases of rules. This CFG also defines two recursive production rules: r_1 determines the rules (RL) included in the knowledge base, and r_3 the clauses (CL) in each rule. G_{RBS} includes a higher number of terminal symbols than G_{DFFNN} (13 vs. 2), which increases the size of the search space and hinders the search for an optimal solution. Finally, G_{SR} shapes the expressions of a symbolic regression optimization problem with two recursive productions, r_1 and r_2, to create unary and binary operations, respectively. Moreover, r_2 is a doubly recursive production rule, which considerably increases the complexity of the search space.

These three benchmark problems have been used in other research to solve real-world problems. G_{DFFNN} is successfully employed in neuroevolution to approximate some sequences related to the theory of orthogonal polynomials (Barrios Rolanía et al.,

Table 3: CFG that define the problems under study.

$G_{DFFNN} = (V, \Sigma, R, L)$	$G_{RBS} = (V, \Sigma, R, RBS)$	$G_{SR} = (V, \Sigma, R, E)$
$V = \{L, N\}$	$V = \{RBS, R, A, C, CL, EV, OP, VL, EQ, VA\}$	$V = \{E, OP1, OP2\}$
$\Sigma = \{/, n\}$	$\Sigma = \{if, then, =, != , \&, \Vert, v1, v2, v3, v4, t, f\}$	$\Sigma = \{+, -, /, *, sin, cos, v\}$
$R = \{$	$R = \{$	$R = \{$
$r_1 : L ::= N \,/\, L$	$r_1 : RBS ::= R\ RBS \mid R$	$r_1 : E ::= OP1\ E$
$r_2 : L ::= N$	$r_2 : R ::= if\ A\ then\ C$	$r_2 : E ::= OP2\ E\ E$
$r_3 : N ::= n\ N$	$r_3 : A ::= CL\ A \mid CL$	$r_3 : E ::= v$
$r_4 : N ::= n\ \}$	$r_4 : CL ::= EV\ OP\ EV$	$r_4 : OP1 ::= sin \mid cos$
	$r_5 : EV ::= VA\ EQ\ VL$	$r_5 : OP2 ::= + \mid - \mid / \mid *\ \}$
	$r_6 : VA ::= v1 \mid v2 \mid v3 \mid v4$	
	$r_7 : EQ ::= != \mid =$	
	$r_8 : OP ::= \Vert \mid \&$	
	$r_9 : VL ::= t \mid f$	
	$r_{10} : C ::= v1 \mid v2 \mid v3 \mid v4\ \}$	

2018). Usual applications include the interpolation and approximation of functions, as well as the construction of quadrature formulas and other problems related to numerical integration. Font et al. (2010) apply a more specific version of G_{RBS} for the evolutionary construction of self-adapting knowledge-based systems for the detection of knee injuries. Finally, Ramos Criado (2017) and Ramos Criado et al. (2020) employ G_{SR} as a benchmark problem to highlight the limitations of GGGP when dealing with large search spaces. It is important to note that no hierarchical optimization has been performed in order to avoid bias from, for example, learning the neural network parameters in the G_{DFFNN} problem.

4.1 Hyperparameter Setup

The first stage of the experimental results aims to tune the hyperparameters of the three GGGP approaches to compare WX, EDA, and SEDA. The replacement rate is one of the hyperparameters under study, since it appears to influence the GGGP performance. Rates of 20%, 40%, 60%, 80%, and 100% are considered. The smoothing rate σ is the other hyperparameter tuned in this stage for SEDA, since it is one of the primary contributions of the proposed approach. Values of 10^{-4}, 10^{-3}, 10^{-2}, and 10^{-1} are tested to cover both high values, which increase SEDA exploration ability, and values close to zero, which lead the evolutionary process towards local search. $\sigma = 0$ means that the offspring is always generated following the CFGE branches already traversed by the parents. It is the usual strategy in general EDA, which is also adopted in the experiments conducted with GGGP-EDA. Therefore, $\sigma = 0$ is not considered in the SEDA-smoothing-factor tuning process.

Statistical analysis was run to compare the fitness of the final solutions using different hyperparameter configurations on each evolutionary approach. The problem involves searching for known specific target derivation trees, each of which encodes the optimal solution. A different target derivation tree with 50 recursions is randomly generated uniformly in each execution and CFG under study: G_{DFFNN}, G_{RBS}, and G_{SR}. Since these grammars define two recursive production rules, the target derivation trees could contain any combination of the two recursive production rules that total

Table 4: Configurations employed in the experiments.

Hyperparameter	Value
GGGP approaches	WX, EDA, and SEDA
Population size	100
Initialization algorithm	Grammatically uniform
Selection operator	Tournament of size 5
Replacement	Substitutes the worst individuals with the offspring
Mutation	None
Replacement rates	20%, 40%, 60%, 80%, and 100%
SEDA smoothing factors σ	10^{-4}, 10^{-3}, 10^{-2}, and 10^{-1}
Fitness function	Levenshtein distance (3)
Executions run	100 for each hyperparameters setup and GGGP approach
Target derivation tree size	50 recursions
Target derivation trees	The same 100 for each hyperparameters setup and GGGP approach
Stop criteria	1. The target derivation tree is found. 2. The avg. population fitness improves less than 1% in 25 evaluations.

50 recursions. For example, in G_{DFFNN}, a target derivation tree may have either 50 recursions of the nonterminal L, N, or a mix of both with 50 recursions in total. For the three evolutionary approaches, the fitness function $f(s, s')$ calculates the Levenshtein distance from the word $s = a_1a_2 \ldots a_n$ encoded in the target derivation tree to the sentence $s' = a'_1, a'_2 \ldots a'_m$ encoded by the individual to be evaluated as defined in (3).

There are 5 possible configurations for the replacement rate using WX or EDA, while there are 20 combinations of replacement rates and smoothing factors for SEDA. For each hyperparameter setup, 100 executions have been performed. Any evolutionary process stops when either the target derivation tree is found or the average population fitness improves less than 1% after 25 evaluations. The same 100 target derivation trees are employed for all evolutionary algorithms and hyperparameter configurations to avoid biases. Table 4 summarizes the configurations employed in the experiments.

The hyperparameter configuration served as the independent variable for each evolutionary algorithm and CFG, with 5 levels in the case of WX and EDA, corresponding to the 5 replacement rates studied, and 20 for SEDA, considering all combinations of replacement rates and smoothing factors. The dependent variable was the fitness of the final solution (the best fitness in the last generation) achieved by each of the three evolutionary approaches using G_{DFFNN}, G_{RBS}, and G_{SR}, respectively.

A one-way between-groups analysis of variance (ANOVA) was conducted to gather empirical evidence of whether the differences between the final fitness means achieved by each evolutionary algorithm and CFG are statistically significant for the different hyperparameter setups considered. One of the conditions of an ANOVA is that the variances of the groups should be equivalent. ANOVA is robust to this violation when the groups are of an equal or near-equal size. This condition holds for the current study since 100 executions were run for all CFG, evolutionary algorithm, and hyperparameter configurations ($N = 100$). The size of the groups also allows the assumption of normality.

The ANOVA test results for WX and EDA working with G_{DFFNN} reveal that the null hypothesis (the means of the best fitnesses achieved by the evolutionary algorithm using G_{DFFNN} to search for a specific derivation tree with 50 recursions are equal) cannot be rejected because of the resulting $F(df = 4/495) = 0.623247$ for WX and $F(df = 4/495) = 0.194756$ for EDA, both $p \geq 0.05$. These two results indicate that the final solutions achieved by GGGP-WX and -EDA provide a statistically similar average fitness when varying the replacement rate. Instead, the null hypothesis can be rejected for SEDA since $F(df = 19/1980) = 111.7893$, $p < 0.05$. Thus, the quality of the solutions that SEDA achieves in this case depends on the hyperparameter replacement rate and smoothing factor.

Similarly, data regarding the ANOVA test results for the three evolutionary algorithms were gathered using G_{RBS} and G_{SR}. The results achieved working with G_{RBS} reveal statistically significant differences between the means of the fitness of the solutions achieved by WX when varying the replacement rate because of the resulting $F(df = 4/495) = 28.04572801$, $p < 0.05$. The same occurs using SEDA with different replacement rates and smoothing factors, $F(df = 19/1980) = 648.3582176$ and $p < 0.05$. However, the null hypothesis cannot be rejected for EDA working with G_{RBS}. In the case of G_{SR}, the ANOVA tests reported sizeable differences for the three evolutionary algorithms: $F(df = 4/495) = 14.20826237$, $F(df = 4/495) = 3.628594826$, and $F(df = 19/1980) = 115.066862$ for WX, EDA, and SEDA, respectively, resulting in $p < 0.05$ for the three of them.

When there were sizeable differences between the groups regarding the hyperparameters, the Tukey HSD (honestly significant difference) test was used to make *post hoc* comparisons to demonstrate where the statistically significant differences are found. This is the case of SEDA with G_{DFFNN}, WX and SEDA working with G_{RBS}, and the three evolutionary algorithms with G_{SR}.

Tables 5 and 6 summarize the data gathered by showing the hyperparameter configurations included in the homogeneous subset for $\alpha = 0.05$ with the best fitness means for each evolutionary algorithm and grammar pair. Table 5 corresponds to SEDA working with G_{DFFNN}, WX with G_{RBS}, and SEDA with G_{RBS}, while Table 6 refers to the three evolutionary algorithms working with G_{SR}. An algorithm does not appear in any column in both Tables 5 and 6 when the obtained solutions quality (fitness) do not statistically depend on its hyperparameters. This is the case of WX and EDA working with G_{DFFNN} and EDA with G_{RBS}.

It is important to note that the evolutionary algorithms increasingly depend on their hyperparameters as the problems become more challenging. The mean values are greater for G_{SR} than for G_{RBS}, and the latter greater than G_{DFFNN}. However, SEDA is robust to change, meaning that the replacement rate chosen or the problem at hand play no part as long as the smoothing rates of 10^{-2} and 10^{-3} are employed, since they are the most repeated values in the first and third columns of Table 5 for G_{DFFNN} and G_{RBS}, and the third column of Table 6 for G_{SR}. Likewise, EDA is also robust to change. It is the most robust algorithm in terms of replacement rate variation; it achieves comparable results for G_{DFFNN} and G_{RBS}, regardless of the replacement rate chosen, and in the case of G_{SR}, it achieves comparable results for replacement rates ranging from 20% to 80%.

This first stage of the experimentation aims to discover the best performing hyperparameter set with each evolutionary algorithm to obtain fair comparative results in the second and third phases. Tables 5 and 6 together allow this goal to be achieved by choosing, for each evolutionary algorithm, those hyperparameters in the subsets'

Table 5: Post hoc test: hyperparameter configurations included in the homogeneous subset ($\alpha = 0.05$) with the best fitness means for SEDA with G_{DFFNN}, WX with G_{RBS}, and SEDA with G_{RBS}. Tukey HSD test. Dependent variable: fitness of the final solution. 100 executions are run in all cases. Asterisks in brackets mark the hyperparameters chosen in the subsequent performance comparisons.

SEDA with G_{DFFNN}		WX with G_{RBS}		SEDA with G_{RBS}	
Hyperparams.	Mean	Hyperparams.	Mean	Hyperparams.	Mean
100%, 10^{-2}	0.39	100% (*)	63.90	100%, 10^{-3}	15.12
80%, 10^{-2} (*)	0.49	80%	65.88	80%, 10^{-2} (*)	16.69
60%, 10^{-2}	0.57	60%	70.58	80%, 10^{-3}	17.02
40%, 10^{-2}	0.73			60%, 10^{-2}	17.65
20%, 10^{-2}	0.85			60%, 10^{-3}	18.77
40%, 10^{-3}	1.29			40%, 10^{-3}	19.15
100%, 10^{-3}	1.32			40%, 10^{-2}	22.33
80%, 10^{-3}	1.44				
80%, 10^{-1}	1.49				
40%, 10^{-1}	1.59				
60%, 10^{-1}	1.66				
60%, 10^{-3}	1.85				
20%, 10^{-3}	1.97				
100%, 10^{-4}	2.02				

Table 6: Post hoc test: hyperparameter configurations included in the homogeneous subset ($\alpha = 0.05$) with the best fitness means for the three evolutionary approaches working with G_{SR}. Tukey HSD test. Dependent variable: fitness of the final solution. 100 executions are run in all cases. Asterisks in brackets mark the hyperparameters chosen in the subsequent performance comparisons.

WX with G_{SR}		EDA with G_{SR}		SEDA with G_{SR}	
Hyperparams.	Mean	Hyperparams.	Mean	Hyperparams.	Mean
100% (*)	99.27	80% (*)	116.26	80%, 10^{-2} (*)	97.77
80%	102.17	60%	116.88	60%, 10^{-2}	98.54
		40%	117.67	80%, 10^{-3}	99.60
		20%	118.62	100%, 10^{-3}	99.79

intersection corresponding to the three CFGs. Thus, the optimal hyperparameters are not problem-dependent. In the case of WX, the best performing replacement rates for the three grammars are 80% and 100%. These values appear in both the second column in Table 5 and Table 6, but WX is not on the first column in Table 5, meaning that the replacement rate does not affect the quality of the solutions achieved with G_{DFFNN}. With EDA, either 20%, 40%, 60%, or 80% perform the best with the three CFGs. Finally, in the case of SEDA, the hyperparameter pairs (replacement rate, smoothing factor) that yield the best results with three grammars are (80%, 10^{-2}), (60%, 10^{-2}), (80%, 10^{-3}), and (100%, 10^{-3}).

From the results shown in these tables, it is possible to obtain a set of hyperparameters that is the same for each evolutionary algorithm in the three grammars employed in the study. In the case of WX, any replacement rate is suitable for G_{DFFNN}, and 100% achieves the best average results in G_{RBS} and G_{SR}. Therefore, 100% is the replacement rate chosen for WX to carry out the performance comparison in stages two and three. EDA perfomance is statistically equivalent for any replacement rate with G_{DFFNN} and G_{RBS}. Additionally, it achieves statistically better results with any replacement rate other than 100% with G_{SR}. 80% is the replacement rate chosen for EDA since it yields the best (although equivalent) average results. Finally, from the four pairs that statistically yield the best means with SEDA for the three CFGs, (80%, 10^{-2}) is chosen because it provides the best and the second-best average results.

In short, for the subsequent comparison stages, WX will employ a generational replacement of 100%, 80% in the case of EDA and SEDA, and $\sigma = 0.01$ as the smoothing factor in SEDA. These configurations are marked with an asterisk in brackets in Tables 5 and 6.

4.2 Performance Comparison

This stage aims to show and compare the three GGGP approaches performance when trying to obtain specific derivation trees of different sizes for the three CFGs of Table 3. The search problem is the same as defined in the hyperparameter setup stage but extended to 10, 20, 50, 75, and 100 recursions to observe each evolutionary algorithm behavior when dealing with small and large derivation trees and search spaces. The experiments conducted employ the same configurations reported in the previous stage and summarized in Table 4, except for the target derivation tree sizes (now extended), the replacement rate, and the SEDA smoothing factor. In this study, the replacement rate adopted is 100% for WX, 80% for EDA and SEDA, and $\sigma = 0.01$ for SEDA as obtained from the hyperparameter setup stage.

The experiments performed in this stage are not concerned with each problem's semantics, such as searching for the best multilayer perceptron or knowledge base of rules to perform a specific task. Instead, these ad-hoc experiments are intentionally provided to avoid the noise produced by other optimization processes, such as learning the parameters of each multilayer perceptron generated during the evolution process. They aim to facilitate revealing the advantages of the three GGGP algorithms and their limitations, especially when dealing with large search spaces.

Comparisons between the three GGGP approaches have been performed in terms of the fitness of the final solutions and convergence speed. Two pairs of plots, one for the final fitnesses and the other for convergence speed, present the statistical results for each of the three CFGs studied. Each pair shares the abscissa axis, representing the number of recursions of the target derivation trees to be found. The chart below in each pair shows descriptive statistics for each target derivation tree size through box-and-whisker plots. The ordinate axis corresponds to the fitness of the final solutions in a logarithmic scale or the number of evaluations performed by each evolutionary algorithm until meeting one of the stop criteria (Table 4). The chart above in each pair shows in the ordinate axis the significance level p achieved by the three one-way analysis of variance (ANOVA) tests on two groups (t-test) conducted for each target derivation tree size to compare the three evolutionary algorithms: EDA vs. SEDA, EDA vs. WX, and SEDA vs. WX. The dependent variable is the fitness of the final solutions when comparing the quality of the solutions achieved or the number of evaluations performed when comparing the

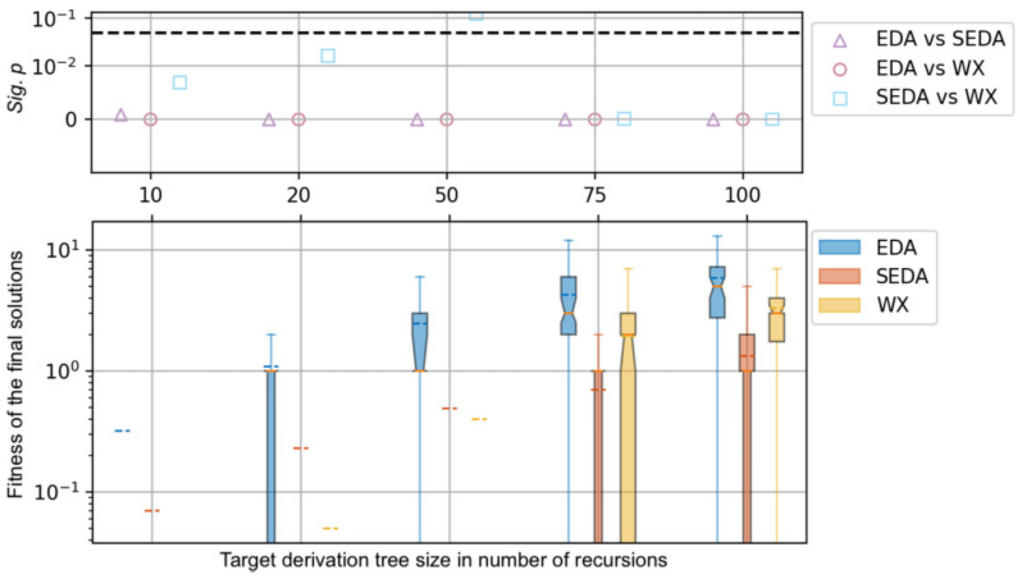


Figure 3: Statistical comparisons in terms of the fitness of the final solutions for the G_{DFFNN} problem after running 100 executions for each evolutionary algorithm and target derivation tree size. Above, the significance levels p achieved by each of the three one-way ANOVA tests on two groups (t-test) performed to compare the three evolutionary approaches. The dashed line represents the significance threshold level $p = 0.05$. Below, the box-and-whisker plots.

convergence speed. In both cases, the independent variable is the evolutionary algorithm employed for each target derivation tree size and CFG.

The aim is to study the impact of the evolutionary approaches on the fitness of the final solutions achieved and the number of evaluations performed for shallow and deep target derivation trees using different CFGs by gathering empirical evidence of whether the differences between the means of the dependent variable are statistically significant. Again, the null hypothesis (the means of the dependent variable are equal for the different independent variable values) is rejected when a significance level of $p < 0.05$ is achieved. A dashed horizontal line represents this significance threshold level in the plot above in each pair of graphs. All groups are of the same size, with 100 observations for each target derivation tree size, evolutionary algorithm, and CFG. Therefore, neither Levene nor Shapiro-Wilk tests were considered.

Figure 3 shows statistical results comparing the three pairs of GGGP approaches in terms of the average fitness of the final solutions achieved for the G_{DFFNN} problem. The ANOVA tests reach a significance level of $p < 0.05$ when comparing EDA with either WX and SEDA, represented by circles and triangles, respectively. These results mean that the differences observed in the box-and-whisker plots below are statistically significant. Figure 3 and the fourth row in Table 7 reveal that EDA generally obtains the worst final solutions. SEDA also achieves significant differences with WX for 10, 20, 75, and 100 recursions (represented by squares in Figure 3). Table 7 reports that WX statistically achieves the best solutions for 10 and 20 recursions, but SEDA prevails over both WX and EDA as the target derivation tree sizes get larger: 75 and 100 recursions.

Table 7: Average fitness of the final solutions and average evaluations, in thousands, performed to meet a stop criterion using G_{DFFNN} after running 100 executions for each evolutionary algorithm and target derivation tree size. Significant best values are marked with an asterisk in brackets.

	10 recursions			20 recursions			50 recursions			75 recursions			100 recursions	
WX	EDA	SEDA	WX	EDA	SEDA	WX	EDA	SEDA	WX	EDA	SEDA	WX	EDA	SEDA
Average fitness of the final solutions														
$0.0^{(*)}$	0.32	0.07	$0.05^{(*)}$	1.09	0.23	0.4	2.47	0.49	1.87	4.25	$0.99^{(*)}$	3.3	5.87	$1.33^{(*)}$
Average evaluations performed (in thousands)														
$0.26^{(*)}$	0.79	0.5	0.98	1.79	1.21	3.97	2.81	2.56	4.79	2.91	3.1	6.21	$2.95^{(*)}$	3.56

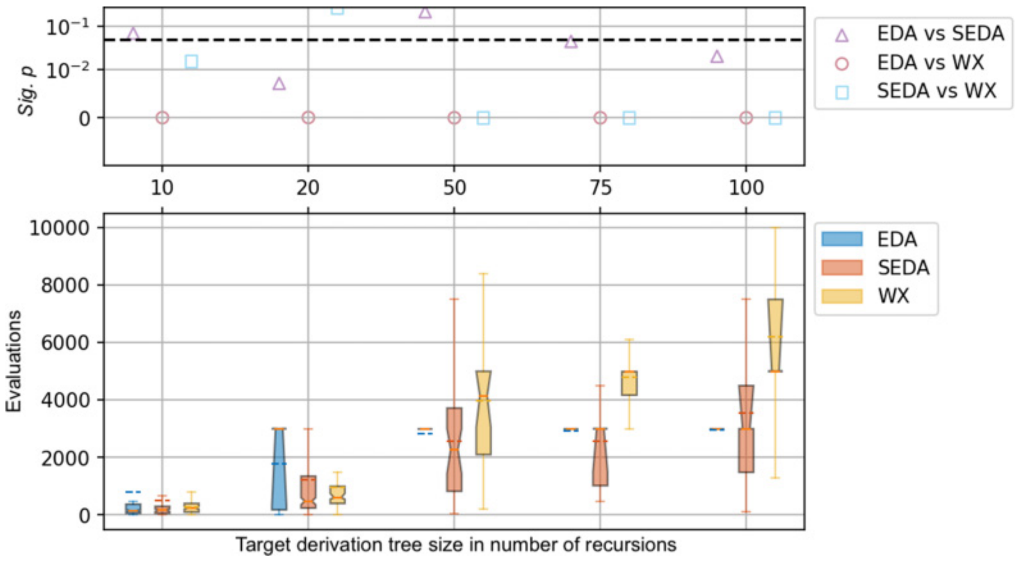


Figure 4: Statistical comparisons in terms of the number of evaluations needed to meet a stop criterion for the G_{DFFNN} problem after running 100 executions for each evolutionary algorithm and target derivation tree size. Above, the significance levels p achieved by each of the three one-way ANOVA tests on two groups (t-test) performed to compare the three evolutionary approaches. The dashed line represents the significance threshold level $p = 0.05$. Below, the box-and-whisker plots.

Figure 4 shows statistical results comparing the three pairs of GGGP approaches in terms of evaluations performed to meet a stop criterion using G_{DFFNN}. Furthermore, the last row of Table 7 shows the average values. The expected number of evaluations increases with the target derivation tree size for the three evolutionary algorithms. The ANOVA tests reach a significance of $p < 0.05$ when comparing WX and EDA, represented by circles in the graph above. Consequently, there are statistically significant differences in the average number of evaluations performed by each algorithm. When the target derivation tree size is up to 20 recursions, WX meets a stop criterion faster than EDA. However, for deeper target derivation trees of 50, 75, and 100 recursions,

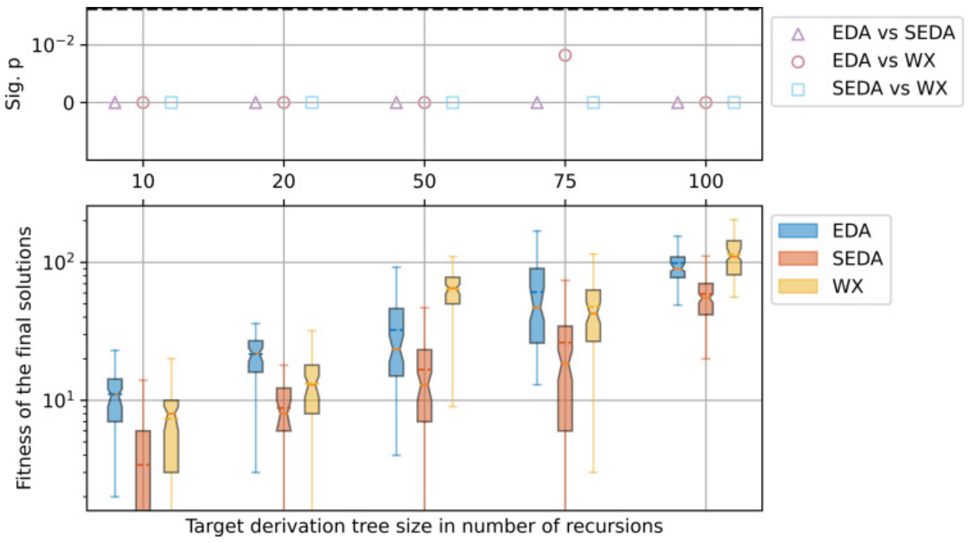


Figure 5: Statistical comparisons in terms of the fitness of the final solutions for the G_{RBS} problem after running 100 executions for each evolutionary algorithm and target derivation tree size. Above, the significance levels p achieved by each of the three one-way ANOVA tests on two groups (t-test) performed to compare the three evolutionary approaches. The dashed line represents the significance threshold level $p = 0.05$. Below, the box-and-whisker plots.

EDA significantly performs fewer evaluations than WX, although it achieves the worst solutions. SEDA always maintains an intermediate average number of evaluations that generally is statistically equivalent to either WX or EDA.

Three main conclusions can be deduced from the results obtained with G_{DFFNN}. First, WX rapidly meets a stop criterion and achieves fair solutions for target derivation trees up to 20 recursions. WX performs a more exploratory search than the other two algorithms, which proves to be suitable for small search spaces. However, as the target derivation trees and, therefore, the search space get larger, EDA performs the fewest number of evaluations, although it stops in poor local optima. Unlike WX, EDA boosts the local search, resulting in it stopping prematurely, which becomes even more pronounced as the problem becomes more challenging. Finally, SEDA, which tries to balance the exploration and exploitation, takes an intermediate number of evaluations to stop and improve the quality of the solutions, compared to WX and EDA, as the target derivation tree sizes increase, being the best of the three for 75 and 100 recursions.

G_{RBS} is a recursive grammar that encodes rule knowledge bases of variable size in the form `if Antecedent then Consequent`. The `Antecedent` of a rule may include one or more clauses. An interesting feature of this grammar is that it can generate broad, not only infinitely deep, derivation trees by applying the recursive production rules r_1 and r_3. This feature makes the problem even more difficult because the words to compare for fitness calculation (target and candidate solutions) are long. Consequently, the evolutionary algorithms take the highest number of evaluations of the three CFGs to reach a stop criterion.

Figure 5 shows the statistical results comparing the three pairs of GGGP approaches in terms of the fitness of the final solutions. All the ANOVA tests reach a significance

Table 8: Average fitness of the final solutions and average evaluations, in thousands, performed to meet a stop criterion using G_{RBS} after running 100 executions for each evolutionary algorithm and target derivation tree size. Significant best values are marked with an asterisk in brackets.

Average fitness of the final solutions														
10 recursions			20 recursions			50 recursions			75 recursions			100 recursions		
WX	EDA	SEDA	WX	EDA	SEDA	WX	EDA	SEDA	WX	EDA	SEDA	WX	EDA	SEDA
7.34	11.1	3.4(*)	13.2	21.64	8.81(*)	63.9	32.4	16.7(*)	47.8	60.9	26.25(*)	114.8	98.77	59.13(*)
Average evaluations performed (in thousands)														
8.7	3.4	3.3	12.1	4.5(*)	5.4	17.5	4.7(*)	8.3	20.3	5.5(*)	9.1	18.8	5.9(*)	10.4

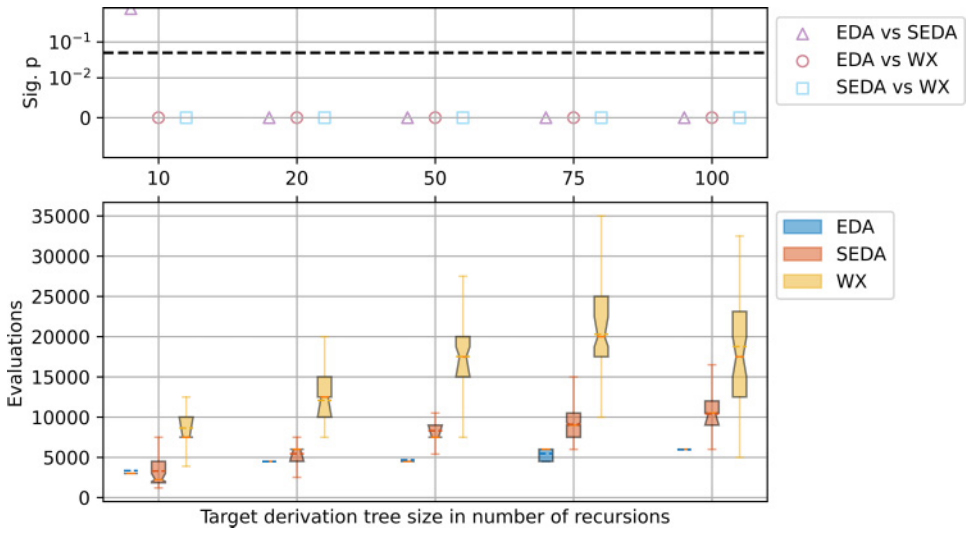


Figure 6: Statistical comparisons in terms of the number of evaluations needed to meet a stop criterion for the G_{RBS} problem after running 100 executions for each evolutionary algorithm and target derivation tree size. Above, the significance levels p achieved by each of the three one-way ANOVA tests on two groups (t-test) performed to compare the three evolutionary approaches. The dashed line represents the significance threshold level $p = 0.05$. Below, the box-and-whisker plots.

level of $p < 0.05$. Therefore, the null hypothesis (the average fitness of the final solutions are equal) can always be rejected. There are statistically significant differences between the average fitness of the solutions provided by each GGGP approach, resulting in SEDA always obtaining the best solutions and EDA obtaining the worst except for 100 recursions. Table 8 shows the exact average fitness values of each evolutionary algorithm and target derivation tree recursions in the fourth row. Note that all the values corresponding to SEDA are marked with an asterisk, which means they are statistically the best of the three algorithms.

Figure 6 shows the statistical results regarding the number of evaluations needed to meet a stop criterion. WX significantly takes more generations to stop for all the target

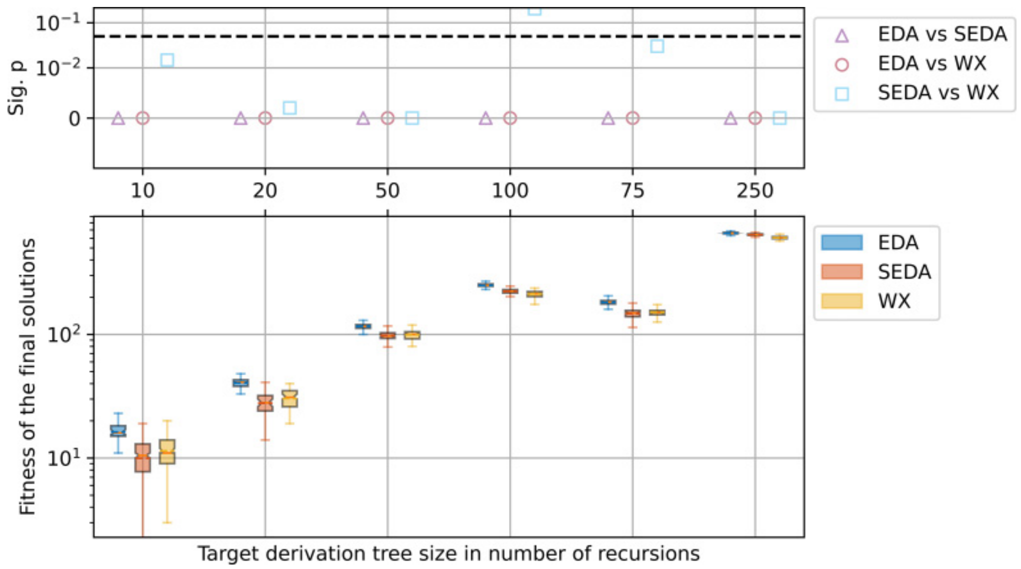


Figure 7: Statistical comparisons in terms of the fitness of the final solutions for the G_{SR} problem after running 100 executions for each evolutionary algorithm and target derivation tree size. Above, the significance levels p achieved by each of the three one-way ANOVA tests on two groups (t-test) performed to compare the three evolutionary approaches. The dashed line represents the significance threshold level $p = 0.05$. Below, the box-and-whisker plots.

derivation tree recursions. EDA is the fastest for 20 to 100 recursions. Still, it achieves the worst quality solutions except for 100 recursions, suggesting an excessive local search capability that prematurely stops the evolutionary process. SEDA takes an intermediate average number of evaluations to achieve the best solutions. The last row of Table 8 is also worth observing, since it shows that SEDA only increases 2.1 thousand evaluations on average, from 8.3 to 10.4, when searching for target derivation trees from 50 to 100 recursions. In contrast, it requires five thousand more evaluations on average to go from 10 to 50 recursions. This result reveals that SEDA performs better as the size of the search space increases working with G_{RBS}.

The challenge with the G_{SR} problem lies in the fact that this grammar includes the double recursive production rule r_2, besides the recursive r_1. The evolutionary algorithms achieve the worst solutions of the three CFGs.

Figure 7 shows the statistical results regarding the fitness of the final solutions. The chart above points out that there are always statistically significant differences in the average fitness between the three pairs of evolutionary algorithms for all the target derivation tree recursions. These results, from the box-and-whisker plot below, and Table 9 indicate that SEDA provides better solutions than WX and EDA approaches except for 100 recursions, while EDA always achieves the worst results.

Figure 8 shows statistical results regarding the number of evaluations performed by the three evolutionary approaches to reach a stop condition for each target derivation tree recursion. Again, the chart above points out statistically significant differences in the average number of evaluations between the three pairs of algorithms. EDA needs the fewest evaluations to stop, and SEDA achieves intermediate results. However, it is

Table 9: Average fitness of the final solutions and average evaluations, in thousands, performed to meet a stop criterion using G_{SR} after running 100 executions for each evolutionary algorithm and target derivation tree size. Significant best values are marked with an asterisk in brackets.

Average fitness of the final solutions														
10 recursions			20 recursions			50 recursions			75 recursions			100 recursions		
WX	EDA	SEDA	WX	EDA	SEDA	WX	EDA	SEDA	WX	EDA	SEDA	WX	EDA	SEDA
11.4	16.2	9.96$^{(*)}$	30.3	40.5	27.8$^{(*)}$	99.3	116.3	97.8$^{(*)}$	151.2	181.9	147.4$^{(*)}$	212.2$^{(*)}$	249.1	223.3
Average evaluations performed (in thousands)														
6.7	3.03$^{(*)}$	4.7	7.2	3.1$^{(*)}$	5.5	8.2	3.1$^{(*)}$	7.4	7.9	3.1$^{(*)}$	7.1	7.6	3.1$^{(*)}$	5.4

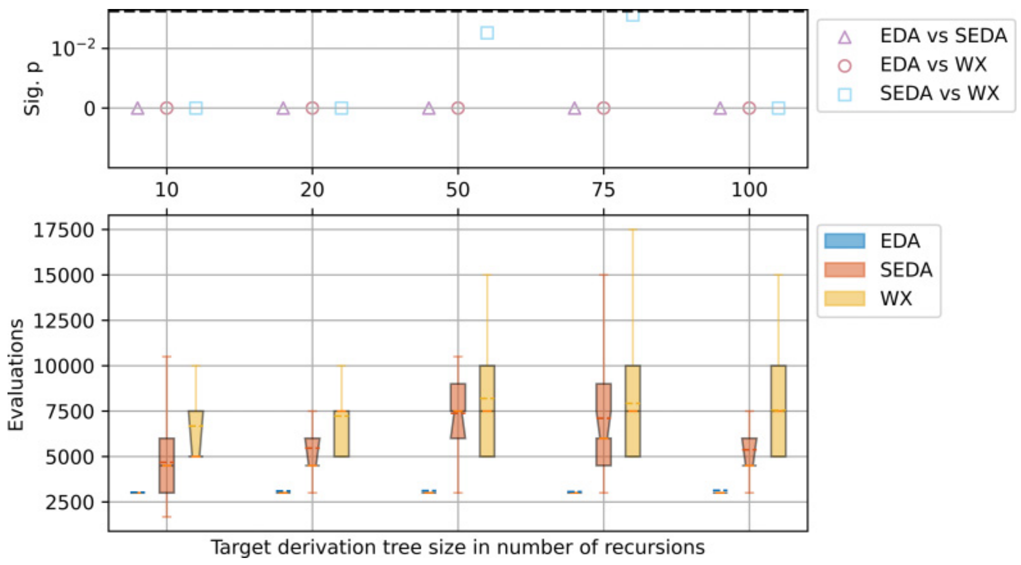


Figure 8: Statistical comparisons in terms of the number of evaluations needed to meet a stop criterion for the G_{SR} problem after running 100 executions for each evolutionary algorithm and target derivation tree size. Above, the significance levels p achieved by each of the three one-way ANOVA tests on two groups (t-test) performed to compare the three evolutionary approaches. The dashed line represents the significance threshold level $p = 0.05$. Below, the box-and-whisker plots.

also worth noting that, from the results reported in Table 9, EDA requires approximately the same number of evaluations for all recursions, about 3.1 thousand. It is clear that EDA displays premature convergence in these experiments: it requires the fewest evaluations to stop but achieves the worst average solutions. On the contrary, SEDA generally achieves the best solutions with an intermediate number of evaluations, suggesting an adequate trade-off between exploration and local search. Finally, WX achieves fairly good solutions using G_{SR}, close to SEDA and prevailing over it for 100 recursions, but it requires the highest number of evaluations, the typical behavior of an excessively exploratory search algorithm.

The results achieved regarding the average fitness of the final solutions and the number of evaluations to meet a stop criterion reveal that the search problem defined by G_{DFFNN} is more simple to solve than G_{RBS} and G_{SR}. According to Tables 7, 8, and 9, the average fitness of the final solutions achieved by the three evolutionary approaches are within the range [0, 5.87] for G_{DFFNN}, [3.4, 114.8] for G_{RBS}, and [9.96, 249.1] for G_{SR}. Moreover, the same tables report that the average evaluations performed to stop are within [0.26, 6.21] thousands for G_{DFFNN}, [3.3, 20.3] thousands for G_{RBS}, and [3.03, 8.2] thousands for G_{SR}.

Additional tests were performed with the mutation operator, the only operator commonly used in GGGP that is not present in the experiments reported, using mutation probabilities of 0.01 and 0.05. After generating the offspring, if the mutation probability states that the operator is applied to one of the offspring individuals (derivation trees), it randomly chooses a node with a nonterminal symbol and replaces the subtree beginning from that node with another one randomly generated by the grammatically uniform initialization method (Table 4).

However, the results achieved with both mutation probabilities were statistically similar to those already reported without the mutation operator. EDA slightly improves (decreases) the average fitness of the final solutions only when the mutation probability is 0.05, with almost no increase in the average number of evaluations needed to meet a stop criterion. Nevertheless, this enhancement is not statistically significant.

The interpretation of these results is that WX is a very explorative variation operator; therefore, adding the mutation has no effect on the evolutionary process. SEDA includes the smoothing factor to control the exploration and local search capabilities; thus, adding a mutation operator has a similar effect to increase this parameter. Finally, GGGP with EDA boosts the local search, which might lead the algorithm to prematurely converge to local optima. In this case, the mutation operator can benefit the evolutionary process by increasing the exploration of the search space towards new individuals that allow it to escape from local optima.

5 Conclusions

This work presents a smoothed estimation of distribution algorithm (SEDA) for grammar-guided genetic programming that provides a trade-off between exploration and local search to efficiently guide the evolutionary process to accurate solutions. The proposed approach calculates the context-free grammar expansion (CFGE) to generate the offspring. The CFGE is a tree-like graph representation of all derivation trees that can be generated by the CFG, together with a probabilistic model of the whole search space that approximates the current population distribution. SEDA implements a code-bloat control mechanism by setting up a recursion bound in recursive CFG to limit the CFGE size. Derivations larger than the CFGE size cannot be generated.

However, under such exploitative conditions, SEDA would prematurely stop. The evolutionary process would be rapidly guided to converge to an optimum in very few generations, but likely to a local optimum. To overcome this difficulty, SEDA *smooths* the estimated distribution model; thus, new individuals may be generated from the current population distribution. These new out-of-distribution individuals increase the population diversity and allow the evolutionary process to explore new promising search space areas.

Statistical results gathered in terms of the average fitness of the final solutions achieved by EDA, WX, and SEDA reveal that EDA finds the worst solutions, showing an excessive local search capability that leads the evolutionary process to prematurely stop. On the other hand, WX achieves the best solutions only with G_{DFFNN} and small search spaces in general, where it is more likely to find a good solution by exploration. This kind of tree-swapping variation operator may be highly exploratory and barely exploitative. Therefore, they are not suitable for problems involving large search spaces with large derivation trees. Finally, SEDA significantly achieves the best solutions on the most challenging problems: G_{RBS}, G_{SR}, and G_{DFFNN} searching for target derivation tree sizes of 75 and 100 recursions.

Regarding the number of evaluations to meet a stop criterion, EDA significantly takes fewer generations to stop in general, showing few exceptions for some of the most straightforward experiments. Therefore, EDA is the fastest algorithm, although it also finds the worst solutions. This behavior is usual in algorithms that boost local search excessively, as EDA does. In contrast, SEDA obtains intermediate results. It is slower than EDA but faster than WX and generally finds the best quality solutions.

The experimental results conducted lead us to assume that SEDA is the most balanced algorithm. While WX is the slowest approach according to its excessive exploration capability, EDA meets a stop criterion very early, getting the worst quality solutions. Alternatively, the hyperparameter smoothing factor regulates the SEDA exploitation–exploration trade-off; the higher the more similar to WX (explorative), the lower the more similar to a regular EDA behavior (exploitative). From the results reported, a smoothing factor of 0.01 makes SEDA balance exploration and local search, achieving the best quality solutions, especially in the most challenging problems, with an intermediate average speed.

This research work shows the difficulties of finding the proper balance between exploratory and exploitative behavior. Although SEDA presents a fit exploitation–exploration trade-off, the elaboration of new specific mutation operators may be beneficial to the overall evolutionary process when working with large search space problems. Such mutation operators must focus on the search of unexplored search space areas, avoiding the re-exploration of already visited areas. A controlled exploration balance between spatially close and distant areas, according to the state of the evolutionary process, may also be beneficial.

New evolutionary optimization techniques may also arise from the presented work. The CFGE represents the search space in a graph-like structure that directly encodes derivation trees, which can be obtained by traversing the CFGE. This search space model can be modified to represent, in addition to trees, any other more complex graph-encoded search spaces. Hence, a new evolutionary optimization technique may be developed to optimize any graph-like problems using regular evolutionary operators.

Upper and lower probability thresholds can be helpful for GP and GGGP EDA approaches, where probabilities can easily reach values 0 and 1. Nevertheless, the side effects of these bounds on the evolutionary process still need to be fully understood. As EDA bounds are only enforced when exceeded, further studies should investigate the potential biases that may arise due to inconsistent shaping of the probability distribution.

Acknowledgments

This work was partially supported by research grant PID2021-122154NB-100 of Ministerio de Economía, Industria y Competitividad, Spain.

The authors thank the reviewers and editors for their valuable comments and suggestions, which have improved this paper.

References

Bäck, T., Fogel, D., and Michalewicz, Z. (1997). *Handbook of evolutionary computation*. Taylor & Francis.

Barrios Rolanía, D., Delgado Martínez, G., and Manrique, D. (2018). Multilayered neural architectures evolution for computing sequences of orthogonal polynomials. *Annals of Mathematics and Artificial Intelligence*, 84(3–4):161–184.

Couchet, J., Manrique, D., Rios, J., and Rodriguez-Paton, A. (2007). Crossover and mutation operators for grammar-guided genetic programming. *Soft Computing*, 11(10):943–955. 10.1007/s00500-006-0144-9

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.

Dorigo, M., Birattari, M., and Stutzle, T. (2006). Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39. 10.1109/MCI.2006.329691

Font, J. M., Manrique, D., and Pascua, E. (2011). Grammar-guided evolutionary construction of Bayesian networks. In J. M. Ferrández, J. R. Álvarez Sánchez, F. de la Paz, and F. J. Toledo (Eds.), *Foundations on natural and artificial computation*, pp. 60–69. Berlin: Springer.

Font, J. M., Manrique, D., and Ríos, J. (2010). Evolutionary construction and adaptation of intelligent systems. *Expert Systems with Applications*, 37(12):7711–7720. 10.1016/j.eswa.2010.04.070

Galván, E., Trujillo, L., McDermott, J., and Kattan, A. (2013). Locality in continuous fitness-valued cases and genetic programming difficulty. In Schütze, O., et al. (Eds.), *EVOLVE—A bridge between probability, set oriented numerics, and evolutionary computation II. Advances in Intelligent Systems and Computing, Vol. 175* (pp. 41–56). Berlin: Springer. 10.1007/978-3-642-31519-0_3

Galván-López, E., McDermott, J., O'Neill, M., and Brabazon, A. (2011). Defining locality as a problem difficulty measure in genetic programming. *Genetic Programming and Evolvable Machines*, 12(4):365–401. 10.1007/s10710-011-9136-3

Galván-López, E., O'Neill, M., and Brabazon, A. (2009). Towards understanding the effects of locality in GP. In *Eighth Mexican International Conference on Artificial Intelligence*. 10.1109/MICAI.2009.17

García-Arnau, M., Manrique, D., Ríos, J., and Rodríguez-Patón, A. (2007). Initialization method for grammar-guided genetic programming. *Knowledge-Based Systems*, 20(2):127–133. 10.1016/j.knosys.2006.11.006

Hasegawa, Y., and Iba, H. (2008). A Bayesian network approach to program generation. *IEEE Transactions on Evolutionary Computation*, 12(6):750–764. 10.1109/TEVC.2008.915999

Hauschild, M., and Pelikan, M. (2011). An introduction and survey of estimation of distribution algorithms. *Swarm Evolutionary Computation*, 1(3):111–128. 10.1016/j.swevo.2011.08.003

Hemberg, M., O'Reilly, U.-M., Menges, A., Jonas, K., Gonçalves, M. d. C., and Fuchs, S. R. (2008). *Genr8: Architects' experience with an emergent design tool*, pp. 167–188. Berlin: Springer.

Hopcroft, J., Motwani, R., and Ullman, J. D. (2006). *Introduction to automata theory, languages and computation*. 3rd ed., Addison-Wesley Longman Publishing.

Joshi, A., and Schabes, Y. (1997). *Handbook of formal languages 3: Beyond words*. Springer.

Kari, L., and Rozenberg, G. (2008). The many facets of natural computing. *Communications of the ACM*, 51(10):72–83. 10.1145/1400181.1400200

Kim, K., and McKay, R. (2013). Stochastic diversity loss and scalability in estimation of distribution genetic programming. *IEEE Transactions on Evolutionary Computation*, 17(3):301–320. 10.1109/TEVC.2012.2196521

Kim, K., Shan, Y., Nguyen, X., and McKay, R. (2014). Probabilistic model building in genetic programming: A critical review. *Genetic Programming and Evolvable Machines*, 15(2):115–167. 10.1007/s10710-013-9205-x

Koza, J. (1992). *Genetic programming: On the programming of computers by means of natural selection*. MIT Press.

Koza, J. R., Keane, M. A., Streeter, M. J., Mydlowec, W., Yu, J., and Lanza, G. (2006). *Genetic programming IV: Routine human-competitive machine intelligence*, Vol. 5. Springer.

Krithivasan, K. (2009). *Introduction to formal languages, automata theory and computation*. Pearson Education.

Looks, M., Goertzel, B., and Pennachin, C. (2005). Learning computer programs with the Bayesian optimization algorithm. In *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 747–748.

McKay, B., Hao, H. T., Mori, N., Hoai, N. X., and Essam, D. (2006). Model-building with interpolated temporal data. *Ecological Informatics*, 1(3):259–268. 10.1016/j.ecoinf.2006.02.005

McKay, R., Hoai, N., Whigham, P., Shan, Y., and O'Neill, M. (2010). Grammar-based genetic programming: A survey. *Genetic Programming and Evolvable Machines*, 11(3–4):365–396. 10.1007/s10710-010-9109-y

Moll, R. N., Arbib, M. A., and Kfoury, A. J. (2012). *An introduction to formal language theory*. Springer Science & Business Media.

O'Neill, M., and Brabazon, A. (2006). Grammatical swarm: The generation of programs by social programming. *Natural Computing*, 5(4):443–462. 10.1007/s11047-006-9007-7

O'Neill, M., and Ryan, C. (2003). *Grammatical evolution: Evolutionary automatic programming in an arbitrary language*. Springer.

O'Neill, M., Vanneschi, L., Gustafson, S., and Banzhaf, W. (2010). Open issues in genetic programming. *Genetic Programming and Evolvable Machines*, 11(3):339–363.

Poli, R., Langdon, W., McPhee, N., and Koza, J. (2008). *A field guide to genetic programming*. Lulu.com.

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.

Ramos Criado, P., Barrios Rolanía, D., Manrique, D., and Serrano, E. (2020). Grammatically uniform population initialization for grammar-guided genetic programming. *Soft Computing*, 24:11265–11282. 10.1007/s00500-020-05061-w

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

Ratle, A., and Sebag, M. (2002). A novel approach to machine discovery: Genetic programming and stochastic grammars. In *International Conference on Inductive Logic Programming*, pp. 207–222. Springer.

Ryan, C., Collins, J., and O'Neill, M. (1998). Grammatical evolution: Evolving programs for an arbitrary language. In *European Conference on Genetic Programming*.

Salustowicz, R., and Schmidhuber, J. (1997). Probabilistic incremental program evolution. *Evolutionary Computation*, 5(2):123–141. 10.1162/evco.1997.5.2.123

Sastry, K., and Goldberg, D. E. (2006). *Probabilistic model building and competent genetic programming*, pp. 205–220. Berlin: Springer.

Shan, Y., McKay, R., Essam, D., and Abbass, H. (2003). *A survey of probabilistic model building genetic programming*, pp. 121–160. Springer.

Sipser, M. (2013). *Introduction to the theory of computation*, 3rd ed. Cengage Learning.

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 the Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 155–166.

Tetteh, M., Dias, D. M., and Ryan, C. (2022). Grammatical evolution of complex digital circuits in SystemVerilog. *SN Computer Science*, 3(188). 10.1007/s42979-022-01045-9

Uy, N., Hoai, N., O'Neill, M., and McKay, B. (2010). The role of syntactic and semantic locality of crossover in genetic programming. In *11th International Conference on Parallel Problem Solving from Nature, Part II*.

Vanyi, R., and Zvada, S. (2003). Avoiding syntactically incorrect individuals via parameterized operators applied on derivation trees. In *Congress on Evolutionary Computation*, Vol. 4.

Whigham, P. (1995). Grammatically-based genetic programming. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*.

White, D., McDermott, J., Castelli, M., Manzoni, L., Goldman, B., Kronberger, G., Jaśkowski, W., O'Reilly, U., and Luke, S. (2013). Better GP benchmarks: Community survey results and proposals. *Genetic Programming and Evolvable Machines*, 14(1):3–29. 10.1007/s10710-012-9177-2

Wong, M. L., and Leung, K. S. (2000). *Data mining using grammar based genetic programming and applications*, Vol. 3. Springer.