Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Departamento de Inteligencia Artificial

New techniques for Grammar Guided Genetic Programming: dealing with large derivation trees and high cardinality terminal symbol sets.

PhD Thesis

Author:

Pablo Ramos Criado
Ingeniero Informático

Supervisors:

Dr. Daniel Manrique Gamo

Dr. José María Font Fernández

Madrid 2017

Propuesta de tribunal aprobado por la comisión de doctorado de la Universidad Politécnica de Madrid, el día ___ de _____ de 2017

Presidente/a: _____

Vocal: _____

Vocal: _____

Vocal: _____

Secretario/a: _____

Suplente: _____

Suplente: _____

Realizado el acto de defensa y lectura de la Tesis el día ___ de _____ de 2017 en la Escuela Técnica Superior de Ingenieros Informáticos

Calificación: _____

PRESIDENTE/A VOCAL 1 VOCAL2

VOCAL 3 SECRETARIO/A

A mi padre

Agradecimientos

La tesis doctoral, tan sencilla en sus extremos, tan tortuosa e intrincada en el camino. Ha sido un largo viaje, con sus más y sus menos, sus alegrías y sus complicaciones y sus vueltas y vueltas, pero todo llega a su fin, y con la misma ilusión que empezaba este proyecto, lo termino contemplando el futuro que llega. Y cómo no, miro atrás y veo el camino recorrido, un camino repleto de personas que han estado conmigo, apoyándome, animándome, o simplemente acompañándome con una sonrisa o unas palabras amables. Quisiera dedicar estás palabras a todas estas personas que tanto han hecho por mí.

Empezaré por donde todo empezó, por aquellas personas que me ayudaron a iniciar este incierto sendero. Gracias Martita, gracias mamá, sin vuestro apoyo y cariño incondicional este sueño no habría sido posible. Gracias, papá, que aún sin estar presente, has sido guía en cada paso.

También quisiera agradecer a mis hermanos, cuñados y padres adoptivos todo el apoyo y cariño que he recibido durante todo este tiempo. A los peques, que aún sin saberlo, me han hecho más fácil mis días de trabajo. Y en general a toda mi familia y amigos que han estado a mi lado todo este tiempo.

Si he conseguido llegar hasta este momento, es gracias a mis directores, Daniel y José, que me han apoyado y guiado durante toda la tesis. Gracias por confiar en mí y por haberme dado esta oportunidad. También a Alfonso y Dolores, cuyo apoyo y conocimiento tanto me han facilitado este viaje. Asimismo, agradezco a Dr. Woolsey la oportunidad de realizar mi estancia de investigación en una universidad tan prestigiosa como Virginia Tech.

Finalmente, y no menos importante, quiero agradecer a todos mis compañeros de laboratorio el cariño y las alegrías que habéis aportado a cada día de mi vida durante estos últimos años. Especialmente a Martín y a Willy, que después de tantos días trabajo, jugos y siestas, os habéis convertido en una parte imprescindible de mi vida.

Table of contents

Abstract

Metaheuristic algorithms have shown an important role in the resolution of complex search and optimization problems. Grammar guided genetic programming is a metaheuristic optimization technique within natural computing that works with solutions of variable size. However, although it is a metaheuristic method, it has shown some limitations when solving problems using context free grammars that generate large search spaces. Concretely, when they generate large derivation trees or possess a high cardinality terminal symbol set.

The presented research work has determined the population initialization and the variation operators as the main causes of these limitations. Population initialization methods are unable to generate a representative sample of the search space. The variation operators are either very explorative or too destructive, being unable to actively guide the optimization process. To address these limitations, three major contributions are presented: a grammatically uniform population initialization, an estimation of distribution crossover operator and an asynchronous endosymbiotic co-optimization technique designed to work with high cardinality terminal symbol sets.

The performed experiments show the advantages of using these novel techniques when solving problems with the above-mentioned characteristics. To such aim, laboratory experiments have been designed to highlight the limitations of the current GGGP methods. Moreover, experiments have been also successfully performed with large search spaced problems involving search and optimization of different types of intelligent systems.

Resumen

Los algoritmos metaheurísticos han demostrado tener un importante papel en la resolución de problemas de búsqueda y optimización complejos. La programación genética guiada por gramáticas es una técnica de optimización metaheurística, enmarcada dentro del campo de la computación natural, que permite manejar soluciones de dimensión variable. No obstante, aunque es un método metaheurístico, muestra ciertas limitaciones en la resolución de problemas con gramáticas libre de contexto que generan espacios de búsqueda grandes. Concretamente, cuando se generan árboles de derivación grandes o el conjunto de símbolos terminales tiene una alta cardinalidad.

El trabajo de investigación presentado en esta tesis muestra que la inicialización de poblaciones y los operadores de modificación son las principales causas de estas limitaciones. Los métodos de inicialización de poblaciones existentes no generan generan muestras representativas del espacio de búsqueda. Por su parte, los operadores de modificación son altamente exploratorios o demasiado destructivos, por lo que no son capaces de guiar el proceso de optimización de forma adecuada. Con el fin de disminuir o resolver el efecto de estas limitaciones se presentan tres contribuciones principales: un método de inicialización de poblaciones gramaticalmente uniforme, un operador de cruce basado en la estimación de distribución de poblaciones y un método de optimización cooperativa asíncrono que permite trabajar con gramáticas con una alta cardinalidad del conjunto de símbolos terminales.

Los experimentos realizados muestran las ventajas de utilizar estas técnicas originales en la resolución de problemas con las características antes mencionadas. Se han diseñado problemas de laboratorio con el fin de mostrar las limitaciones de las técnicas actuales en GGGP. Asimismo, se han llevado a cabo experimentos de forma satisfactoria con problemas cuyos espacios de búsqueda son grandes para la búsqueda y optimización de distintos tipos sistemas inteligentes.

I. INTRODUCTION

1. Introduction

Natural Computing is a subfield of Artificial Intelligence that involves two research branches: creating new models and computational techniques inspired by nature and attempting to understand nature in terms of information processing (Kari & Rozenberg, 2008). Evolutionary Computation (EC) (Kari & Rozenberg, 2008) (Bäck, et al., 1997) is a subfield of Natural Computing that borrows ideas from natural evolution (Darwin, 1859) to perform search and optimization processes to populations of individuals (search space) that represent candidate solutions to a specific problem (solution space). Genetic Algorithms (GA) (Goldberg & Holland, 1988) (Holland, 1992), Genetic Programing (GP) (Koza, 1992) and Grammar Guided Genetic Programming (GGGP) (Whigham, 1995) are techniques that belong to EC, namely, evolutionary algorithms (EA). GAs employ fixed-size vectors (individuals) which encode possible solutions of a problem. The size of the vector is established at the start of the algorithm, so that it is not possible to find solutions encoded in vectors of a different size. On the contrary, GP employ programs of variable size so that it is possible to find solutions of different sizes. Grammar guided Genetic Programming (Whigham, 1995) is an evolutionary optimization technique based on GP designed to optimize programs belonging to a search space defined by a Context Free Grammar (CFG) (Hopcroft, et al., 2013) (Sipser, 2012) (Krithivasan, 2009) (Moll, et al., 1988). The CFG defines a set of syntactical restrictions for each problem that all individuals (in this particular case, derivation trees) have to comply with.

The evolutionary optimization process of GA, GP and GGGP comprises three major stages: the population initialization, the fitness evaluation and the population update. The population initialization generates the first set of individuals that encode possible solutions to the problem at hand. The fitness evaluation measures the goodness of the encoded solutions. Finally, the population update performs the optimization process. The population update comprises three major operations: the selection of promising individuals, the production of new

individuals encoding characteristics of the promising solutions by the so-called variation operator, and the replacement of poorly-fit individuals (Font, et al., 2009).

Like other EAs, GP and GGGP start generating the initial population. The performance of these evolutionary algorithms is related to how representative of the search space the initial population is (Burke, et al., 2004). The initial population is usually generated at random (Koza, et al., 2006) (Mckay, et al., 2010) with the goal of obtaining a uniformly distributed sample that improves the overall evolutionary performance. However, random not always means uniform and the initialization process should be driven to artificially achieve representative population samples. Additionally, there are some constraints to consider during initialization that prevent some common issues of GP and GGGP. A common issue is the generation of individuals that do not comply with problem restrictions, called infeasible individuals. The constraint that prevents producing infeasible individuals is called closure property (Mckay, et al., 2010) (Poli, et al., 2008). Several approaches have been presented to comply with closure property during GP initialization. However, they usually include ad-hoc constraints that entangle the GP initialization. GGGP is a variant of GP designed to ensure the closure property throughout all the evolutionary process. A CFG is used to delimit GP search space and drive evolutionary process (Whigham, 1995). During the initialization process, individuals (derivation trees) are randomly generated using the provided CFG to achieve the feasibility. The code bloat is another common issue on GP and GGGP EAs which leads the evolutionary process to disproportionately large solutions (dal Piccol Sotto & de Melo, 2016). Size bounds settings have been postulated as a simple solution to reduce the code bloat in GP (Poli, et al., 2008) and GGGP (Mckay, et al., 2010). Bound settings may benefit the evolutionary process by limiting code bloat. GP showed the importance of defining size bounds on population initialization. Inherited from GP, GGGP has also included these practices from the very beginning (Whigham, 1995) (Gruau, 1996) (Whigham, 1995).

Several approaches that claim more uniform population initialization and reduce code bloat have been presented for GP (Poli, et al., 2008) (Luke & Panait, 2001) (Böhm & Geyer-Schulz, 1996) (Iba, 1996) (Langdon, 2000) (Luke, 2000) (Chellapilla, 1997) (Trujillo, et al., 2016) (Alonso & Schott, 2013) and GGGP (Whigham, 1995) (Gruau, 1996) (Whigham, 1995) (García-Arnau, et al., 2007) (Ratle & Sebag, 2000) (Ványi & Zvada, 2003). Moreover, there are also some GP and GGGP variants that pursues a more uniform initial population (Ratle & Sebag, 2002) (Ratle & Sebag, 2001) (Tanev, 2004) (Murphy, et al., 2012) (Vanneschi, et al., 2014). Most of them, provide methods that bias population initialization to some specific individual sizes that improve uniformity. Depth and breadth of derivation trees have been a target of uniformity search. GGGP approaches deal with inherent CFG additional constraints that make obtaining a specific distribution more difficult. Therefore, the CFG must be considered to design a process that generates uniform populations. Every CFG possesses different production rules and symbols that bias population initialization process in different ways. Some GGGP population initialization approaches analyze the CFG characteristics to describe how derivation trees are produced and guide the population initialization process. However, analyzing CFG biases is complex and initial populations are not usually uniform. The GGGP research has also focused on including size bounds to reduce code bloat. However, size bounds can be inappropriate for GGGP, independently of the CFG type, either non-recursive or recursive. On the one hand, non-recursive CFGs already provide some search space delimitations that may be sufficient to reduce the code bloat. On the other hand, the recursive CFG search space is infinite and size bounds are usually applied to delimit the search space. Even though, size bounds are not suitable since they are not strictly related to the search space or how derivation trees grow. In fact, different depth bounds may not vary the set of derivation trees within the search space.

Once the population is initialized, the evolutionary process starts. The evolutionary process comprises three main operations: selection, variation and replacement. The

selection operation chooses some promising individuals of the population, called the parents. The variation operation generates a set of new individuals based on the characteristics of the selected individuals, called the offspring. Then the replacement operator insert these new individuals into the population and removes other less promising individuals. The selection and variation operators are crucial to guide the evolutionary process towards promising areas of the search space. The performance of variation operators relies on the exploitation-exploration trade-off (Mckay, et al., 2010). On the one hand, the exploitation focuses on the characteristics of parent individuals to produce new individuals with similar characteristics. On the other hand, the exploration produces less related new individuals that are based on parents' characteristics. The crossover and mutation are two variation operators designed for GGGP (Whigham, 1995). The WX has been widely tested and it has shown a good exploitation-exploration trade-off that in general achieves satisfactory results (Mckay, et al., 2010). However, there is still margin for improvement and GGGP has been criticized for being a very restrictive environment where new variation operators are hardly designed (Mckay, et al., 2010). There are two main research lines focused on addressing these limitations. Firstly, the linearization of CFG derivation trees (O'Neil & Ryan, 2003) and the replacement of CFGs by tree adjoining grammars (Joshi & Schabes, 1997) has been proposed to change the search encoding scheme in order to provide an enhanced environment where new variations operators can be design. Secondly, other promising optimization techniques such as estimation of distribution algorithms (EDA) (Larrañaga & Lozano, 2001) or ant-colony optimization algorithms (ACO) (Dorigo, et al., 2006) have been also proposed as GGGP optimization methods. However, there are still limitations related to the exploration-exploitation trade-off of variation operators.

In addition to these limitations, the variation operators present important limitations when optimizing problems with high cardinality terminal symbol set. When the cardinality of the terminal symbols set is higher than the cardinality of non-terminal symbols set, the GGGP performance decreases. For example, when

the terminals are natural or real numbers. These limitations have been addressed by combining different optimization techniques that collaborate to reduce each other disadvantages. Nature has shown how long-term relationships between different organisms can be reciprocally beneficial as well as an important fact for the survival of those organisms. These relationships between two or more different biological species are called symbiosis (Bary, 1879). When there is a symbiosis between an organism and another organism that hosts the first one, the symbiosis is known as endosymbiosis. Both, symbiosis and endosymbiosis, enable a gradual adaptation that, in the long term, may facilitate the creation of better adapted species. This phenomenon is called symbiogenesis (Sapp, et al., 2002). The three of them provide an additional mechanism for adaptation that may have an important role in evolution (Margulis, 1993) (Sapp, et al., 2002).

The phenotypic plasticity is another way of adaptation (DeWitt, et al., 1998): it enables organisms to adapt to their environment all through their lifetime. Although this is a short-term adaptation, James Baldwin postulated an indirect mechanism for eventual inheritances of acquired characteristics that allows to keep these short-term adaptations in the long term. This is known as the Baldwin Effect (Baldwin, 1896) (Morgan, 1896) (Osborn, 1896). A sustained behavior of a species may shape its evolution so that it would evolve into an organism with an increased capacity for learning new skills (Hochberg, 2011). When these learning capacities are widely spread in the population, they might turn into instinctive behaviors that can be inherited (Baldwin, 1896) (Morgan, 1896) (Osborn, 1896). In this regard, learning can be presented as another important factor in evolution. The Lamarckian evolution is another theory that defines a mechanism to inherit these short-term adaptations. This theory asserts that characteristics that have been acquired during lifetime can be directly inherited by the offspring (Burkhardt, 2013). However, this theory has been widely criticized as a nature's evolutionary theory.

These nature concepts have been applied to model multi-level optimization approaches (Heywood & Lichodzijewski, 2010) (Migdalas, et al., 2013), where two

or more optimization techniques hierarchically cooperate to find a solution for a problem at hand. Several multi-level optimization techniques have been presented looking for the benefits of a symbiotic relationship by consecutively (Alonso, et al., 2009) (Panyaworayan & Wuetschner, 2002) (Cagnoni, et al., 2005) or synchronously applying the optimization processes (Couchet, et al., 2007) (Topchy & Punch, 2001) (Keijzer, 2003) (Cagnoni, et al., 2005) (Cerny, et al., 2008) (Mukherjee & Eppstein, 2012) (Vanneschi, et al., 2006). Moreover, other approaches have attempted to overcome EC difficulties by means of learning techniques (Ackley & Littman, 1991) (Downing, 2001) (Hinton & Nowlan, 1987) (Whitley, et al., 1994) (Downing, 2001).

Within this scope, where local optimization techniques are applied to optimize a subset of the terminal symbol set, there are also some limitations related to the definition of an adequate encoding scheme for the terminal symbol subset. General-purpose encoding schemes do not usually consider problem-specific requirements, such as dependent problem constraints, so infeasible (not valid) individuals are often generated (Nowling & Mauch, 2011). A common constraint in optimization problems appears when the sum of the elements that are part of a solution must be a constant, as when working with probabilities (Su, et al., 2008) (Russell & Norvig, 2009). If a general-purpose encoding scheme is chosen, then it is necessary to design a strategy to deal with infeasible individuals. However, when a specific encoding scheme is used, such constraints are already considered (Choubey, 2010).

There are two common ways to deal with infeasible individuals when a general-purpose encoding scheme is employed (Coello, 2016): penalty functions or repair operators. The former approach applies a penalty to the fitness function so that infeasible solutions are allowed, but they are not favored for selection. The main disadvantage of this solution is the premature or slow convergence, as an increasingly number of invalid solutions with poor fitness may exist in the population. This avoids that genetic operators, such as crossover, use their inherent heuristics to guide the evolutionary process to the optimum (Beaser, et al., 2011). Repair operators modify invalid genes to make such individual encodes a feasible

solution. These operators are not usually easy to design and add an undesirable computational cost to the evolutionary algorithm (Dong, et al., 2007).

To overcome the above-mentioned limitations, three major contributions are presented in this thesis: a population initialization that generates grammatically uniform populations, a crossover operator with an enhanced exploitation behavior based on estimation of distributions, and finally an strategy to deal with problems with large cardinality terminal symbols sets together with an encoding scheme for optimizing real numbered vectors, guaranteeing that the vector components add up to some predefined constant.

A grammatically uniform population initialization (GUPI) is presented to improve the initial population uniformity and to provide more suitable bounds that reduce the code bloat. This new approach refers to uniformity as the genotypic or grammatical uniformity: the equiprobability of obtaining any individual of the search space defined by the CFG. Grammatical uniformity also provides a feasible approximation to solution space uniformity. Grammatically uniform populations generally increase phenotypic diversity. If unambiguous CFGs are employed, then there is a one-to-one mapping between the search space and the solution space encoded by the grammar. Then, grammatically uniform populations are also uniform in the solution space. To achieve uniformity, the proposed population initialization is driven by a dynamic stochastic CFG. The probabilities of this stochastic CFG are dynamically assigned to shape population initialization according to the initialization context. The initialization context depends on the CFG characteristics and each derivation tree construction process. Additionally, dynamic recursion bounds are included to control code bloat and drive the initialization to population distributions that improve the overall evolutionary optimization process when recursive CFGs are employed.

A new crossover operator with an enhanced exploitation behavior based on the estimation of distribution is presented. This operator is called estimation of distribution crossover (EDX). The dynamic stochastic CFG presented for GUPI is extended to represent the search space generated by the CFG as a tree graph,

where each node represents a derivation tree node, and the children branches each possible derivation from that node. This graph-like representation of the CFG search space is called CFG expansion. In addition to the probabilities associated to choose a production rule within a nonterminal production set, a new probability is assigned to each branch of the CFG expansion graph. This new probability encodes the relative frequency of a specific derivation within all possible derivations, starting from a specific derivation tree node. The EDX uses the CFG expansion to encode an estimation of distribution of some promising solutions of the population. Then, according to this distribution, a new set of individuals is generated. Although EDX borrows ideas from EDAs, this operator acts as crossover, and produces a small set of individuals from a reduced set of selected promising individuals of the population, instead of producing a new whole population like EDA does.

Inspired by nature's evolution and learning theories, a novel asynchronous symbiotic hierarchical optimization approach is presented to overcome the limitations related to high cardinality terminal symbol sets: the endosymbiotic co-optimization (ECO) approach. Contrary to synchronous approaches, where GP evolution awaits new individuals to be optimized and obtains its fitness and terminals, ECO consists of a GGGP algorithm that implements an embedded local search optimization algorithm (LS) on every GGGP individual (derivation tree). Each embedded LS executes a continuous learning process that concurrently optimizes the derivation tree terminals, improving the fitness of the whole derivation tree. Moreover, estimates of GGGP individuals' fitness are asynchronously provided while derivation tree terminals are being optimized. These fitness estimations guide the metaheuristic process of GGGP evolution, what enables the establishment of an endosymbiotic relationship between the GGGP and the LS that takes advantage of both algorithms benefits. Firstly, the GGGP generates feasible individuals of variable length. Secondly, the LS concurrently optimizes the terminal symbols, avoiding the performance drawback. Optimized terminals obtained by the learning process cannot be immediately inherited. However, bioinspired by Baldwin effect, the optimized characteristics that are

widely spread in the population may be transferred to the derivation trees (genotypes) of GGGP algorithm, making the most of the endosymbiotic relationship.

An alternative real-valued encoding scheme called partition based encoding scheme (PBES) is presented showing to be effective and simple without requiring any modification to the standard numerical optimization techniques. This encoding scheme complies with two important constraints for many real-world problems: the decoding process transforms a genotype into a decoded real-valued vector, guaranteeing that its components belong to a previously fixed interval, and that they exactly add up to the same predefined constant, what for example is applicable to uncertainty reasoning. The decoding process is based on partitioning a real-valued interval into as many segments as the dimension of the decoded solution vector. This encoding scheme is well suited to problems whose solutions have strongly interrelated parts, such as conditional probability tables (Russell & Norvig, 2009), i.e. a bi-dimensional array where every row is a fixed-length vector whose elements add up to 1, and each of them belongs to the interval [0,1].

Empirical experiments have been performed to show the benefits of the proposed methods. Four main set of experiments have been performed: Firstly, a set experiments to show the benefits of GUPI compared to a regular GGGP population initialization in terms of uniformity and GGGP performance when using either the Whigham crossover (WX) or an EDA variation operator approach. Both have been chosen since they are some of the most common variation operators and because of their simplicity and the good results that they achieve. Secondly, a set of experiments to show the performance of EDX compared to the WX and EDA. Thirdly, a set of experiments to show the performance of ECO (asynchronous GGGP + Local search) compared to a plain GGGP and a synchronous GGGP + Local Search optimization approach. In addition, experiments when using ECO with PBES encoding scheme are also conducted. Finally, a set of experiments to show the global performance of GGGP optimization algorithm that comprises GUPI, EDX and ECO, compared

to a synchronous hierarchical optimization approach of GGGP using the regular population initialization together with the WX or EDA are presented. Five different context free grammars encoding the correspondent search spaces of some of most representative GGGP problems are employed in the aforementioned experiments. These problems are: feed-forward neural networks, deep feed-forward neural networks, rule-based systems, symbolic regression and Boolean optimization.

The results achieved show that GUPI obtains better uniform initial populations even for recursive CFG, where the standard stochastic CFG does not properly work. Moreover, the experiments confirm the performance advantages of using better uniform initial populations. The experiments related to EDX show performance improvement in most of the tested scenarios, even when mutation is not applied. The results achieved by ECO show the advantages of using ECO system even when it is compared to a GGGP synchronous approach. Furthermore, including PBES shows the benefits of using a custom encoding scheme that preserves problem restrictions and avoid the generation of infeasible individuals. Finally, the set of experiments related to deep learning problems shows a performance improvement of the proposed methods when dealing with high dimensional problems.

The rest of this document is structured as follows: after this introduction, the state of the art for GGGP is presented. CFGs and evolutionary algorithms are firstly presented to later describe in detail some of the most common GGGP. Then, the research methodology is presented together with the main goals, hypothesis, restrictions and assumptions for the presented research. According to these guidelines, the problem statement describes the main hypothesis and corroborate them by means of theoretical and empirical experimentation. Then, the new techniques designed to address the GGGP limitations are presented together with the empirical results that corroborate their goodness in different scenarios. Finally, the contributions of this research are summarized in the conclusion section and some future research lines are presented.

II. STATE OF THE ART

2. Context-Free Grammars and Languages

Languages can be described as a formal system of symbols (de Saussure, 1992). The set of all symbols that forms a language is known as the alphabet language, $\sum$. Symbols are meaningful by itself, however, a properly structured set of symbols, denoted as string, can provide more complex meanings. The set of all possible strings that can be generated from an alphabet is called the alphabet universe, $W(\sum)$. The subset of strings belonging to a language is called the language over an alphabet, $L(\sum)$. Grammars define systems of rules that generate meaningful combinations of symbols: words or sentences. These grammars are commonly known as formal grammars (Chomsky, 1959). Formal grammars were initially defined in the linguistic field to provide a set of rules that tries to define syntactical restrictions of natural languages (Chomsky, 1959). However, formals grammars were later adopted in other disciplines and nowadays they are also widely used in mathematics and computer science to define formal languages (Hopcroft, et al., 2013) (Sipser, 2012). Formal grammars are defined as a quad-tuple $G = (V, \sum, R, S)$ where:

1. $\sum$ is the alphabet of symbols, called terminal symbols.
2. V is the set of intermediate symbols called variables or nonterminal symbols.
3. $\sum$ and V are disjoint sets, $\sum \cap V = \emptyset$
4. S is the initial nonterminal symbol, called axiom.
5. R is the set of production or rewriting rules that generate the words or sentences.

Terminal symbols are usually represented by lower-case letters near the beginning of the alphabet: a, b, and so on. In the case of nonterminal symbols, upper-case letters near the beginning of the alphabet are used: A, B, and so on. Lower-case letters near the end of the alphabet, such as y and z, denote strings of terminal symbols. For strings consisting of terminals and/or nonterminals symbols, lower case Greek letters, such as α and β, are used (Hopcroft, et al., 2013). Productions rules, $\delta ::= \gamma$, also noted as $r: \delta ::= \gamma$ for the production rule r, consist of two parts:

the head or left-hand side (LHS) of the rule, δ; and the body or right-hand side (RHS) of the rule, γ. The head represents a string $\delta = \alpha A \beta$ where $A \in V;\ \alpha, \beta \in (\sum \cup V)^*$ to be replaced. The body, $\gamma \in (\sum \cup V)^*$, is the string that replaces δ. A rule $r: \delta ::= \gamma$ can be applied to a string φ when it contains the string δ, $\varphi = \alpha\delta\beta$, by replacing δ part of the string by γ, $\alpha\gamma\beta$. Given a nonterminal symbol δ, all the production rules in R of the form $\delta ::= \gamma$ comprise the production rules set, denoted as R_δ. According to the form and restrictions of the production rules, grammar can be classified in 4 types from less restrictive to more restrictive. Grammars of type i are less restrictive than $i + 1$ and $i + 1$ grammars comply at least with i grammars restrictions. This formal language classification its known as Chomsky Hierarchy (Chomsky, 1956) (Chomsky & Schützenberger, 1963):

- **Type 0 (unrestricted):** Production rules can be formulated in any way provided that there is at least one nonterminal symbol on the LHS of the production rule.

$$R = \{\delta ::= \gamma \mid \delta = \alpha A \beta;\ \gamma, \alpha, \beta \in (\sum \cup V)^*;\ A \in V\}$$

- **Type 1 Context sensitive grammar**: Productions rules keep context of nonterminal symbols where rule changes are applied. The LHS and RHS of production rules possess at least one common string. The LHS has higher or equal number of symbols than the LHS, excepting $S ::= \varepsilon$ that, given the axiom, produces an empty sentence ε.

$$R = \{(S ::= \varepsilon), (\alpha A \beta ::= \alpha \gamma \beta) \mid \alpha, \beta \in (\sum \cup V)^*;\ A \in V;\ \gamma \in (\sum \cup V)^+\}$$

- **Type 2 Context free grammar (CFG):** Production rules do not keep context of nonterminal symbols where rule changes are applied. That is, nonterminal symbols do not depend of symbols surrounding them. The LHS of a production rule only has one nonterminal symbol.

$$R = \{(S ::= \varepsilon), (A ::= \gamma) \mid A \in V, \gamma \in (\sum \cup V)^*\}$$

  Languages generated by a CFG are called Context-Free Languages (CFL).

- **Type 3 Regular grammar:** Production rules only generates either one terminal symbol or a string of one nonterminal and one terminal symbol. There are two types of regular grammars depending on the structure of the production rules.
    - **Left regular grammar:**

$$R = \{(S ::= \varepsilon), (A ::= Ba), (A ::= a) \mid A, B \in V;\ a \in \sum\}$$

    - **Right regular grammar:**

$$R = \{(S ::= \varepsilon), (A ::= aB), (A ::= a) \mid A, B \in V;\ a \in \sum\}$$

2.1. CFG Derivations and Properties

Production rules of a CFG are used as rewriting rules to obtain the CFL sentences. Given a CFG $G = (V, \sum, R, S)$, the rewriting process starting from a string $\alpha = \rho A \omega$ with $A \in V$ and $\rho, \omega \in (\sum \cup V)^*$, where a set of n production rules, $\{r^i\}_{i=1}^n$, is applied to obtain a new string $\beta = \rho\gamma\omega$ with $\gamma \in (\sum \cup V)^*$, is denoted as derivation (Hopcroft, et al., 2013) (Sipser, 2012) (Chomsky, 1956). We can formally define the direct or one-step derivation as the binary relation $\Rightarrow$ where, given two strings α, β, we have

$$\alpha \Rightarrow \beta$$

if and only if there exists some production rule $(A ::= \gamma) \in R$ (Krithivasan, 2009). The transitive closure of $\Rightarrow$, where one or more production rules $n \geq 1$ is applied, is noted as $\overset{+}{\Rightarrow}$. The reflexive and transitive closure with $n \geq 0$ is noted as $\overset{*}{\Rightarrow}$. A derivation involving n steps is noted as $\overset{n}{\Rightarrow}$ (Krithivasan, 2009). E.g. this is a derivation using the CFG $G_{a^n b^n} = (\{E\}, \{a, b\}, R, E)$ where $R = \{E ::= aEb,\ E ::= ab\}$.

$$E \overset{n-1}{\Longrightarrow} a^{n-1}Eb^{n-1} \Rightarrow a^{n-1}abb^{n-1} = a^n b^n$$

Any string $\alpha \in (\sum \cup V)^*$ such that $S \overset{*}{\Rightarrow} \alpha$ is named *sentential form* (Moll, et al., 1988). A derivation that produces a terminal string $s \in \sum^*$, $A \overset{+}{\Rightarrow} s$, is called terminal string derivation (Krithivasan, 2009). If A is the axiom S, then $S \overset{+}{\Rightarrow} s$ is a complete derivation, and s is a sentence (Moll, et al., 1988). The set of all sentences derived

from the axiom S is the language of the grammar G: $L(G) = \{s \mid S \overset{*}{\Rightarrow} s;\ s \in \sum^*\}$ (Hopcroft, et al., 2013) (Sipser, 2012) (Chomsky, 1956). Two grammars are equivalent if they generate the same language independently of the derivations. For the given example $G_{a^n b^n}$, any string $a^n E b^n$ is a sentential form, and any derivation $E \overset{n}{\Rightarrow} a^n b^n$ with $n \geq 1$ is a terminal string derivation and also, a complete derivation. The language $L(G_{a^n b^n})$ is $a^n b^n$ with $n \geq 1$.

Derivation steps can be applied in different ways according to the order in which nonterminal symbols are selected (Hopcroft, et al., 2013) (Sipser, 2012) (Krithivasan, 2009). When the leftmost occurrence of a nonterminal is chosen at each step, such derivation is denoted as the leftmost derivation: $\underset{lm}{\Rightarrow}$. In the same way, when the rightmost nonterminal occurrence is selected at each step, it is denoted as the rightmost derivation: $\underset{rm}{\Rightarrow}$. There is no difference between leftmost and rightmost derivations for $G_{a^n b^n}$. However, there are other grammars where the derivation step order makes a difference: $G_{Exp} = (\{E\}, \{1, +, *\}, R, E)$ where $R = \{E ::= E + E,\ E ::= E * E\ E ::= 1\}$. In the following examples, the bold nonterminals are chosen for rewriting. This is a leftmost derivation example:

$$\boldsymbol{E} \underset{lm}{\Rightarrow} \boldsymbol{E} * E \underset{lm}{\Rightarrow} 1 * \boldsymbol{E} \underset{lm}{\Rightarrow} 1 * \boldsymbol{E} + E \underset{lm}{\Rightarrow} 1 * 1 + \boldsymbol{E} \underset{lm}{\Rightarrow} 1 * 1 + 1$$

However, the rightmost derivation that produces the same sentence is the following:

$$\boldsymbol{E} \underset{rm}{\Rightarrow} E * \boldsymbol{E} \underset{rm}{\Rightarrow} E * E + \boldsymbol{E} \underset{rm}{\Rightarrow} E * \boldsymbol{E} + 1 \underset{rm}{\Rightarrow} \boldsymbol{E} * 1 + 1 \underset{rm}{\Rightarrow} 1 * 1 + 1$$

Although derivations are the same, the derivation steps order differs depending on which nonterminal symbol is selected first. The same sequence of production rules applied with both, leftmost and rightmost derivation strategies, produces different derivation for G_{Exp}. This is an example for the following sequence of production rules $(E ::= E * E, E ::= 1, E ::= E + E, E ::= 1, E ::= 1)$

$$\boldsymbol{E} \underset{lm}{\Rightarrow} \boldsymbol{E} * E \underset{lm}{\Rightarrow} 1 * \boldsymbol{E} \underset{lm}{\Rightarrow} 1 * \boldsymbol{E} + E \underset{lm}{\Rightarrow} 1 * 1 + \boldsymbol{E} \underset{lm}{\Rightarrow} 1 * 1 + 1$$

$$\boldsymbol{E} \underset{rm}{\Rightarrow} E * \boldsymbol{E} \underset{rm}{\Rightarrow} \boldsymbol{E} * 1 \underset{rm}{\Rightarrow} E + \boldsymbol{E} * 1 \underset{rm}{\Rightarrow} \boldsymbol{E} + 1 * 1 \underset{rm}{\Rightarrow} 1 + 1 * 1$$

Let $D(s)$ denote the set of all derivation trees for s. A sentence is ambiguous if it can be generated by more than one derivation: $D(s) > 1$. A grammar that contains

at least an ambiguous sentence is also ambiguous. Formally, a CFG $G = (V, \sum, R, S)$ is ambiguous if there exist some string $s \in L(G)$ that has two distinct leftmost (or rightmost) derivations (Hopcroft, et al., 2013) (Krithivasan, 2009). Thus, G is unambiguous if every s has a unique leftmost (or rightmost) derivation. Accordingly, G_{Exp} is ambiguous since there exist two different leftmost derivations that produce the sentence $1 * 1 + 1$:

$$\boldsymbol{E} \underset{lm}{\Rightarrow} \boldsymbol{E} * E \underset{lm}{\Rightarrow} 1 * \boldsymbol{E} \underset{lm}{\Rightarrow} 1 * \boldsymbol{E} + E \underset{lm}{\Rightarrow} 1 * 1 + \boldsymbol{E} \underset{lm}{\Rightarrow} 1 * 1 + 1$$

$$\boldsymbol{E} \underset{lm}{\Rightarrow} \boldsymbol{E} + E \underset{lm}{\Rightarrow} \boldsymbol{E} * E + E \underset{lm}{\Rightarrow} 1 * \boldsymbol{E} + E \underset{lm}{\Rightarrow} 1 * 1 + \boldsymbol{E} \underset{lm}{\Rightarrow} 1 * 1 + 1$$

Any context-free language (CFL) can be produced by infinite CFGs. A CFL L is inherently ambiguous if every CFG G for L is ambiguous. However, if there is at least one unambiguous G that generate L, then L is also unambiguous (Hopcroft, et al., 2013). Ambiguous CFGs have higher cardinality of complete derivations in contrast to the sentences of its CFL. In general, this characteristic does not show any advantage or disadvantage. However, it is possible to surmise that complexity associated to use pairs (G, L) can be higher for those pairs with higher difference in their cardinalities. Hence, unambiguous grammars where every sentence of the language is only generated by one derivation are usually more convenient. Since every unambiguous CFL is generated at least by one unambiguous CFG, it is possible to obtain an unambiguous CFG that generates that language. Given an ambiguous CFG that generates an unambiguous CFL, it is theoretically possible to define a method able to obtain an unambiguous CFG for that CFL (Hopcroft, et al., 2013). However, this is not always computationally feasible. Moreover, sometimes there is not even a method able to decide whether a CFG is ambiguous or not (Hopcroft & Ullman, 1969).

A CFG G is *recursive* (Krithivasan, 2009) if there exist any derivation produced by G such that

$$A \overset{n}{\Rightarrow} \beta \mid \beta = \rho A \omega;\ A \in V;\ \rho, \omega \in (\sum \cup V)^*;\ n \geq 1$$

If $n = 1$, then the recursion is *direct*, and the rule that produce it, A::= $\gamma \mid \gamma = \rho A \omega$, is a recursive production rule. Since recursive derivations can be indefinitely

produced, the language generated by a recursive grammar is infinite. If the grammar is nonrecursive, the languages is finite. Both, $G_{a^n b^n}$ and G_{Exp} are recursive grammars since both produce recursive derivations: $E \overset{n}{\Rightarrow} \beta \mid \beta = \rho E \omega;\ E \in V;\ \rho, \omega \in (\sum \cup V)^*$. Moreover, both have recursive production rules: $E ::= aEb$ for $G_{a^n b^n}$, and $E ::= E + E$ and $E ::= E * E$ for G_{Exp}.

Derivations can be graphically represented as derivation trees, also known as parse trees (Hopcroft, et al., 2013) (Sipser, 2012) (Krithivasan, 2009). A derivation tree is a tree graph where the nodes represent the symbols of a grammar, and the arcs represent the derivations. The root node of a derivation tree is the axiom of a grammar. The leaf nodes are terminal symbols and every other node between the root and the leaves are nonterminal symbols. Derivations are represented by drawing arcs from the axiom or nonterminal symbols to every symbol that they produce. In that way, the parent nodes of a derivation tree represent the nonterminal symbols on the LHS of production rules, while the child nodes represent the terminal and nonterminal symbols on the LHS. Derivation trees can be only used to represent derivations of formal grammars of type 1, or more restrictive type (type 2 and 3) of Chomsky Hierarchy (Chomsky, 1956), that is, context-sensitive grammar, context-free grammar and regular grammars. Figure 1 shows an example of a derivation tree for $\mathrm{E} \overset{+}{\Rightarrow} 1 * 1 + 1$ of G_{Exp}. In this example, the nonterminal symbols and the axiom are represented by non-shadowed nodes and terminal symbols by shadowed nodes. The set of all terminal nodes, taken from the left to the right part of the derivation tree form the sentence or word produced by the derivation tree: $1 * 1 + 1$.

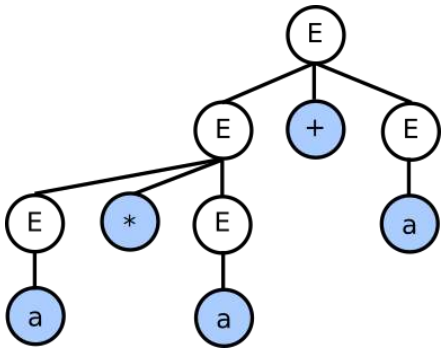

Figure 1: Derivation tree example

2.2. Proper CFGs

Context Free Grammars possess a set features that allow to produce languages which are interesting for computer science (Martin, 2011). However, there is scope for improvement and other properties are usually desirable in computer science. Proper CFGs define some of these desirable properties. A proper CFG is the one that presents no unreachable or unproductive symbols, and that contains no unproductive production rules (ε-production) or unit production rules (Hopcroft, et al., 2013) (Isasi, et al., 1997).

Unreachable symbols are terminal and nonterminal symbols that cannot be reached by any derivation started from the axiom (S):

$$A \in V \mid \nexists S \overset{*}{\Rightarrow} \alpha;\ \alpha = \rho A \omega;\ \alpha, \rho, \omega \in (\sum \cup V)^*$$

$$a \in \sum \mid \nexists S \overset{*}{\Rightarrow} \alpha;\ \alpha = \rho a \omega;\ \alpha, \rho, \omega \in (\sum \cup V)^*$$

The following algorithm detects the existing unreachable symbols in a CFG (Isasi, et al., 1997):

Table 1: Algorithm to detect unreachable symbols

1. *List nonterminal and terminal symbols of the CFG*
2. *Mark the axiom as already found.*
3. *Repeat while there are marked symbols on the list:*
    a. *Choose one marked symbol of the list and delete it.*
    b. *Look for every production rule with that nonterminal symbol on the LHS and mark every symbol on the LHS of the rules. If the marked symbol is a terminal symbol, delete it from the list.*
4. *Every left symbol on the list is an unreachable symbol.*

Unreachable symbols can be removed from a CFG by eliminating them from the nonterminal and terminal symbol sets, and eliminating those production rules with these symbols in any of rule's sides.

Unproductive symbols are nonterminal symbols unable to produce terminal string derivations.

$$A \in V \mid \nexists A \overset{*}{\Rightarrow} s, s \in \sum^*$$

Every production rule that at least contains one unproductive symbol in any of the rule's sides is a superfluous rule. The following algorithm detects the existing unproductive symbols in a CFG (Isasi, et al., 1997):

Table 2: Algorithm to detect unproductive symbols

1. *List nonterminal symbols of the CFG*
2. *Repeat if at least one nonterminal is deleted in step a*
    a. *Delete from the list every nonterminal symbol that appears on the LHS of a production rule with only terminal symbols*
    b. *Delete from the LHS of production rules every non-listed nonterminal symbol.*
3. *Every nonterminal symbol left on the list is an unproductive symbol.*

Unproductive symbols can be avoided by deleting them from the nonterminal symbol set and deleting their corresponding superfluous rules.

If a symbol is neither reachable nor productive, then, it is a useless symbol. Useless symbols must be deleted from CFG since they are not able to produce any sentence.

Unproductive production rules (ε-production) are rules that only produce the empty word (ε) and its left-hand nonterminal symbol is different from the axiom.

$$A ::= \varepsilon \mid A \neq S$$

The following algorithm deletes the unproductive production rules from a CFG (Isasi, et al., 1997):

Table 3: Algorithm to delete unproductive production rules

1. *For every production rule of the type* $A ::= \varepsilon \mid A \neq S$
    a. *Delete the production rule*
    b. *Duplicate every production rule with A on the LHS and delete A symbol from the LHS of the new production rule. If the new rule already exists, delete it. (e.g.* $B ::= \alpha A \beta, B ::= \alpha \beta$*)*

Unit production rules are production rules that only produce a nonterminal symbol.

$$A ::= B \mid A, B \in V$$

Unnecessary production rules are a specific type of unit production rules where LHS is exactly the same as the LHS ($A = A$). These rules can be directly deleted from the CFG since they do not provide any additional information. Sometimes unit production can be useful to create unambiguous grammars. However, unit productions can obfuscate the understanding of CFGs by adding technically unnecessary steps into derivations. To delete unit production rules from a CFG, all unit pairs must be found first. Two nonterminal symbols are a unit pair if there is a derivation such as $A \overset{*}{\Rightarrow} B \mid A, B \in V$. The rest of the algorithm works as described below (Hopcroft, et al., 2013):

Table 4: Algorithm to delete unit production rules

1. *List all unit pairs of the CFG*
2. *For each unit pair* (A, B) *add to* R *a new production rule in the form* $A ::= \alpha$ *for every non-unit production rule* $B ::= \alpha$*.*
3. *Delete all unit production rules from the grammar.*

Note that unnecessary production rules are also deleted by this algorithm, avoiding possible cycles in the CFG derivation.

Every CFG can be transformed into an equivalent proper CFG (Hopcroft, et al., 2013) by following the instructions provided to locate and delete useless symbols, empty and unit productions. To safely transform a CFG into a proper CFG the following sequence must be followed (Hopcroft, et al., 2013):

Table 5: General algorithm to transform a CFG into a proper CFG

1. *Eliminate empty productions.*
2. *Eliminate unit productions.*
3. *Eliminate useless symbols.*

2.3. Stochastic Context Free Grammars

A Stochastic Context Free Grammar (SCFG) is a CFG where a rewriting probability $P(r_i)$ is associated to each production rule $r \in R$ such that $\sum P(r) = 1$ must be complied for every nonterminal production rule set (Giegerich, 2014). SCFGs are defined as a quad-tuple $G = (V, \sum, R, S)$ where V represent the set of nonterminal symbols, $\sum$ the set of terminal symbols, S the axiom, and R a set of probabilities and production rules pairs $(P(r_i), r_i)$. Given a nonterminal symbol $A \in V$, each production rule $r_i: A ::= \alpha \in R$ will be applied according to the associated probability $P(r_i)$.

(1) shows the probability of a complete derivation $S \overset{+}{\Rightarrow} s$, $P\left(S \overset{+}{\Rightarrow} s\right)$. This probability is equivalent to the probability of iteratively applying each of the n production rules in $S \overset{+}{\Rightarrow} s$. P($\{r^i\}_{i=1}^n$). $P(\{r^i\}_{i=1}^n)$ is obtained by multiplying the rewriting probability of each production rule r_i, $P(r_i)$.

$$P\left(S \overset{+}{\Rightarrow} s\right) \equiv P(\{r_i\}_{i=1}^n) = \prod_{i=1}^{n} P(r_i) \qquad (1)$$

Since the product of probabilities is a communicative and an associative operation, $P\left(S \overset{+}{\Rightarrow} s\right)$ can be re-factored as $\prod_{r_i \in \{r_i\}_{i=1}^n} P(r_i)^{n_i}$, where n_i denotes the number of times the production rule r_i is applied in $S \overset{+}{\Rightarrow} s$ (Giegerich, 2014). Since the probability is obtained from a product of probabilities, this value tends to 0 as the number of production rules in $\{r^i\}_{i=1}^n$ increases (Booth & Thompson, 1973). The probability of a sentence s, P(s), equals the sum of the probability of each complete derivation that yields s, $S \overset{+}{\Rightarrow} s$:

$$P(s) = \sum_{\left(S \overset{+}{\Rightarrow} s\right) \in D(s)} P\left(S \overset{+}{\Rightarrow} s\right)$$

SCFGs define a probability distribution on $L(G)$ so that $\sum_{s \in L(G)} P(s) = 1$. If the CFG is unambiguous, then $P\left(S \overset{+}{\Rightarrow} s\right)$ equals the probability of generating the string s, $P(s)$. Let see an example of derivation using the CFG $G_{a^n b^n} = (\{E\}, \{a, b\}, R, E)$ where $R = \{(P(r_1), E ::= aEb),\ (P(r_2), E ::= ab)\}$.

$$E \overset{n-1}{\Longrightarrow} a^{n-1} E b^{n-1} \Rightarrow a^{n-1} a b b^{n-1} = a^n b^n$$

$$P(s) = P\left(S \overset{+}{\Rightarrow} s\right) = P(r_1)^{n-1} P(r_2)$$

Since $G_{a^n b^n}$ is unambiguous $P(s) = P\left(S \overset{+}{\Rightarrow} s\right)$. $E \overset{n}{\Rightarrow} a^n b^n$ applies r_1, $n-1$ times, and r_2 just once, so that $n_1 = n - 1$ and $n_2 = 1$.

3. Evolutionary Computation

Evolutionary computation (EC) is a research subfield of natural computing that borrows ideas from biological evolution to create new optimization methods (Kari & Rozenberg, 2008). Evolutionary computation algorithms are commonly population-based algorithms where an optimization process is applied to the individuals of the population. These population-based optimization algorithms bioinspired by biological evolution are called evolutionary algorithms (EA) (Dasgupta, 2013). There exist several variants of EAs designed to solve a significant amount of different problems methods (Kari & Rozenberg, 2008). EAs have proven its adequacy for solving a wide range of problems where the computational cost is unacceptable when using classical optimization methods. This cost is usually related to the search space evaluation. Since EAs do not make any assumption of the search space and guide the evolutionary process by a heuristic function, they show a good performance approximating solutions to many kind of complex problems.

Conceptually, EAs create an artificial environment where a population of individuals must survive (Kari & Rozenberg, 2008). This environment represents

the problem to solve, and the individuals, candidate solutions to that problem. To survive, the population of individuals must artificially evolve to adapt themselves to the environment. The more adapted individuals survive while the less adapted individuals are rejected. At the end of the evolutionary process, the set of best adapted individuals represent the set of solutions.

The rest of this chapter is organized as follows. In Section 3.1, evolutionary algorithms are firstly introduced. Then, some of the most popular evolutionary algorithms optimizing programs are presented. To provide some scope to programming approaches of EAs, genetic algorithms are introduced in section 3.2. Then genetic programming, grammar guided genetic programming, and other genetic programming variants are described in detail in sections 3.3, 3.4 and 3.5 respectively. To finish, some hybrid evolutionary algorithms, where evolutionary algorithms are combined to improve the optimization performance, are presented in section 3.6.

3.1. Evolutionary Algorithm Definition

Evolutionary algorithms are metaheuristics whose optimization processes are performed by some evolutionary operators that iteratively update a population of individuals until a stop criterion is met (Bäck, et al., 1997). EAs comprise an encoding system, a population initialization method, an evaluation or fitness method, a set of evolutionary operators and a stop criterion. The encoding system provides a mechanism to stablish a relationship between the candidate solutions of a problem and the genotype of the individuals that represent them in the evolutionary process. The population initialization method generates the first set of individuals that form the population. The evaluation or fitness method measures the adaptation to the environment of population individuals, that is, how good the encoded solutions are. The set of evolutionary operators update the population seeking to produce a new more adapted population. The stop criteria define a set of requirements to finish the evolutionary process.

In general terms, an EA works as follows (Font, et al., 2009) (Figure 2): Firstly, a random initial population is generated. The set of individuals representing the population at each iteration is called generation. The fitness evaluation is applied every iteration to measure how adapted the new generation is. If a stop criterion is met, then the evolutionary process stops and returns the current population. If not, the evolutionary operators are applied to update the population. As a result, the population will gradually evolve to better adapted individuals until a stop criterion is met.

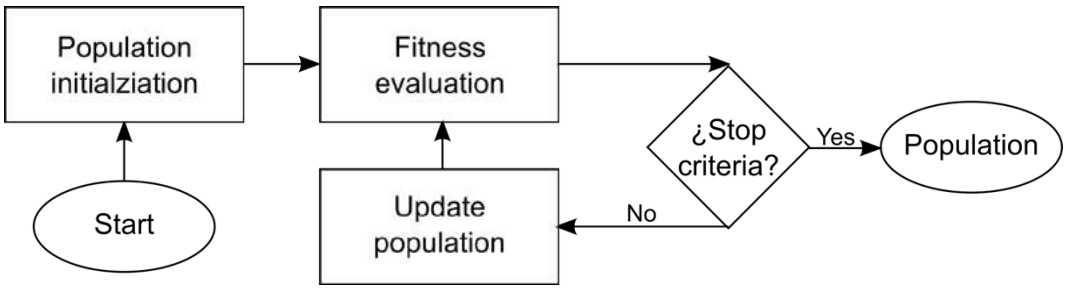

Figure 2: Evolutionary algorithm flow chart

The encoding system is bioinspired by nature's genotype-phenotype mapping. As it occurs in nature, the phenotype is the set of observable characteristics of an individual's solution, while the genotype is the encoded information that determines the characteristic that define the solution (Bäck, et al., 1997). The genotypic information is arranged following some restriction so that evolutionary operators can be applied. The set of all possible genotypes forms the problem search space, while the set of all possible solution, the solution space. The encoding system definition is usually a critical matter in EC. A one-to-one genotype-phenotype mapping is usually a desirable characteristic. More important is that the encoding system provides a genotype for every phenotype of the solution space so that any solution can be found by the optimization process. In the same way, every genotype must be related to a phenotype to avoid invalid solutions appear in the population.

3.1.1. Population Initialization

The population initialization method must provide a representative sample of the search space (García-Arnau, et al., 2007), that is, a diverse set of individuals.

Diversity is the genotypic or phenotypic variety of a population. More diverse populations are usually related to more representative population samples. Since new individuals are usually generated based on previous ones' characteristics, having a diverse population is essential for the optimization process. Thus, more diverse populations improve EAs optimization process while poor diverse populations may produce premature convergence to local optima (Burke, et al., 2004). The population size is usually fixed during all the evolutionary process. The population initialization generates a predefined size set individuals set. Then, every population update must keep the population size. This requirement is usually complied to avoid diversity lose or increasing the computational resources requirements.

3.1.2. Fitness Evaluation

The fitness evaluation guide the optimization process by revealing which individuals may lead to promising solutions. Inadequately defined evaluation methods may reduce EAs performance or even mislead the evolutionary process to improper solutions. According to Darwinian evolution, the individuals more adapted to the environment possess higher probabilities to survive and breed than the less adapted individuals (Engelbrecht, 2007). The evaluation method provides and objective function that measures how close an individual's encoded solution is to solve a given problem, that is, how adapted it is. This function usually returns a number. This number is called fitness. According to the desired fitness value, the EA will perform a maximization or minimization process. When more adapted individuals return low fitness rates, the EA performs a minimization process, otherwise, it performs a maximization. The solution of an individual that minimizes (or maximizes) the fitness function is called an optimal solution or global optimum. Those solutions with fitness values that outscore close solutions in the solution space are called local optima. Metaheuristics usually returns local optima as the optimization process does not ensure finding the optimal solution. EAs present this same issue as they are metaheuristic methods; however, local optima solutions are usually adequate approximations of problem solutions.

3.1.3. Stop Criteria

The fitness function provides a valuable information to the stop criteria. The stop criteria define a set of requirements to finish the evolutionary process. Since the EAs goal is finding the best feasible solution to a given problem, the most common criterion is to stop when the optimal solution is found. However, the solution space is generally unknown and there is no way to determine whether the optimal solution is found or not. In this case, the EA is usually stopped when the population converges. A population converges when its individuals present very similar genotypes or the EA is not able to obtain better solutions. The following criteria can be used to reveal when population converges: best or average population fitness does not improve in a predefined amount of generations, individuals' genotypes are similar or equal, or a predefined number of generations is reached. When one of these criteria is met, the evolutionary process stops and returns the current population.

3.1.4. Evolutionary Operators

Evolutionary operators are also guide by fitness function. These operators provide a heuristic to update the population and lead the optimization process to promising solutions. There are three common evolutionary operators: selection, variation, and replacement. The selection operator chooses some of the best-fit individuals within the population. These individuals are commonly called parents, or variation or crossover pool. The variation operator produce a new set of individuals, called offspring, according to the characteristics of the individuals provided by the selection operator. The replacement selects some of the least-fit population individuals and eliminate them to later introduce the new generated individuals. Variants operators depends on the encoding system and they are designed accordingly. However, selection and replacement operators can be applied to any EA.

Selection Operator

The selection operator guides the evolutionary process by choosing those individuals that may breed more-fit individuals. This operator is bioinspired in Darwinian natural selection (Engelbrecht, 2007). The best-fit individuals to their environment are more likely to survive and breed than the least-fit individuals, whose likely perish or don't succeed in breeding. In the same way, selection operator chooses some of the best-fit individuals to breed. Since the offspring fit is directly dependent on progenitor's genotype, selection operator is a key factor in the evolutionary process. Some of more popular selection operators are presented (Sivaraj & Ravichandran, 2011) (Noraini & Geraghty, 2011):

I. **Tournament selection (TS):** This is probably the most popular selection operator due to its efficiency and simplicity. A set of N individuals are randomly selected from the population. This set is referred as tournament pool and N as the tournament size. The tournament size is usually set to 2. Within the tournament pool individuals, the best-fit individual is selected. This operator preserves diversity since any individual excepting for the least-fit individual can be selected. Accordingly, there is a lower probability to premature convergence or best-fit individuals' dominance. Higher tournament sizes lead to less diverse population since best-fit individuals are more likely selected. Additionally, since tournament pool individuals are randomly selected, there is no need for fitness sorting.

II. **Proportional roulette wheel selection or fitness proportionate selection:** The individuals are proportionally selected according to their fitness values. The probabilities of selecting individuals are represented as segments of a roulette wheel. These probabilities are directly proportional to individuals' fitness. Those individuals with larger fitness are represented by larger segments and consequently, they are more likely selected. The probability p_i of selecting an individual i from a population of size N is defined as,

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$$

where f_j is the fitness of an individual j. The individuals' segments are arranged in the roulette wheel so that each individual is represented by a range with values within the range $[0,1]$. This range is defined by the cumulative sum of the previous arranged individuals' probabilities as the lower range value, and this same cumulative sum plus the individual's probability value as the upper range value. To select an individual, the roulette wheel spinning is simulated by generating a random number within the range $[0,1]$. The individual whose range contains that value is selected. The main advantage of this method is that no individual is discarded during selection. However, very different fitness values may lead to diversity loss and premature convergence. On the contrary, if individuals' fitness values are very similar, the system may not evolve since the selection does not guide the optimization process.

III. **Ranked roulette wheel selection (RRWS):** To deal with proportional RWS issues, ranked RWS is presented. In this case, the probability value is assigned according to the individuals' fitness rank in the entire population rather to their fitness values. A mapping function is applied to assign the probabilities according to the rank. This function can be linear or non-linear (linear ranking or non-linear ranking). An example of linear ranking is presented where probabilities is determined by the constant selection pressure (SP). To obtain the segment size s_i of each individual i according to their rank $rank$, the next formula is used,

$$s_i(rank) = 2 - SP + \left(2(SP-1)\frac{rank-1}{N-1}\right)$$

These values are normalized to obtain the individuals probabilities. This new approach of RWS avoid premature convergence when individuals' fitness values are very different, since probabilities are not assigned according to their fitness values but to their rank. Thus, the dominance of individuals whose fitness value is highly above the average is avoided. Moreover, the evolutionary process is always guided to promising solutions

regardless the similarity of fitness values, since better individuals are always promoted even when there is a just a little difference on fitness values.

IV. **Stochastic universal sampling:** It is a variant of roulette wheel selection operators to avoid selection bias. This variant sample all the individuals by only generating one random number. To select n individuals, the roulette wheel is evenly divided in n parts starting from a random value. The cut points determine the selected individuals.

V. **Elitist selection:** This is the simplest selection operator when population is sorted by fitness. The best individuals of the population are always selected. Despite its simplicity, this operator is not usually applied to generate a new offspring since there is a strong bias towards best-fit individuals that cause population diversity reduction and premature convergence. However, this operator is usually applied to replicate best-fit individuals in the new generation when all individuals of the population are replaced.

Variation Operator

Variation operators produce a new set of individuals according to the characteristics of the individuals provided by the selection operator. These operators are usually applied following a probabilistic distribution. The set of individuals created by the variation operator are usually called offspring. Inspired by nature, some variation operators have been designed to create a reproduction process. There are two types of reproduction: sexual and asexual reproduction. The sexual reproduction creates a new set of individuals by recombining two parents' genomes. The major exponent of sexual reproduction in EA is the crossover operator (Holland, 1992). The asexual reproduction creates new individuals by replicating genomes. Replicating a genome is a simple task in EC. However, this operation does not provide any advantage for itself to the evolutionary process. Thus, little variations must be performed to the genotype to seek more-fit individuals that lead the evolutionary process. This process is called mutation and is performed by the mutation operator (Holland, 1992). The mutation modifies individuals genotype by randomly changing some of its genes. The mutation

provides a tool for exploring new search space areas that may contain new promising solutions. The crossover operator also take advantage of mutation by applying it to the offspring. However, the mutation frequency is a critical matter for the crossover operation. When applying this operator with a low probability, the evolutionary process is favoured by increasing genome diversity. However, an excessive use may be detrimental to the evolutionary process.

There are some EAs that have shown a better performance by combining multiple genomes. These algorithms are not bioinspired in nature's reproduction, but in other nature's patterns like animals or particles behaviour. Behind the bioinspired evolutionary pattern of these algorithms, a distribution representing the characteristic of the promising solutions is encoded, so that each new generation is created based on promising characteristics.

Replacement Operator

The replacement operator provides a tool to introduce the new generation into the population by eliminating some of the least-fit individuals. The number of population individuals that are replaced every generation by the new individuals is determined by the replacement rate $\tau \in (0,1]$. The replacement rate denotes the population percentage of replaced individuals. Depending on the replacement rate value, the evolutionary algorithm is called generational or steady-state evolutionary algorithm. On the one hand, generational algorithms create a new generation of individuals that fully replace the old population. That is, $\tau = 1$. On the other hand, steady-state algorithms use a replacement rate $\tau \in (0,1)$ so that the population is partially replaced by the new generated individuals keeping some of the best-fit individuals. When partially replacing the population, a strategy to select the individuals to be replaced is needed. As selection operator, these strategies usually employ fitness to select the individuals to replace, so that, the least-fit individuals are more likely replaced. Some of the most common strategies are (Engelbrecht, 2007):

I. **Least-fit replacement:** The least-fit individuals are replaced by the new individuals. This is the simplest replacement operator when population is

sorted by fitness. Diversity reduction may appear in small populations since least-fit individuals are always selected.

II. **Random replacement:** Some random individuals are selected independently of their fitness and replaced by the new individuals. Promising or even optimal solutions are replaced with the same probability than other individuals. Thus, evolutionary performance can be drastically reduced.

III. **Elitist replacement:** Individuals are only replaced if the new individuals outperform the fitness of individuals to be replaced (Castelli, et al., 2016). This approach protects promising solutions from being replaced by less-fit individuals. However, this fact can cause a diversity reduction or even halt the evolutionary process.

IV. **Tournament replacement:** The tournament replacement works as selection tournament, but instead of selecting the best-fit individuals of tournaments, the least-fit individual is selected. This is also probably the most popular replacement operator due to its efficiency and simplicity.

V. **Elderly replacement:** Oldest individuals of the population are selected to be replaced by the new individuals. This strategy force the renovation of population increasing population diversity. However, promising or even optimal solutions will eventually be deleted if stop criteria is not met.

VI. **Parent replacement:** New individuals replace the parents that breed them with a certain probability. Since parent are usually more-fit individuals, promising or even optimal solutions may be deleted. However, the offspring is based on parents' genotype. Therefore, when using appropriate crossover and mutation operators, there is no performance drawback.

There are other replacement operator variants designed to replace a set of individuals by some randomly generated new ones. These strategies are intended to increase population diversity and explore new feasible solutions (Font, et al., 2009).

I. **Packaging operator:** All duplicated individuals or individuals with the same fit but one are eliminated. This strategy is very common since it increases population diversity avoiding side effects.

II. **The last judgment:** All population individuals but the best-fit one are eliminated. This operator is usually employ to drastically improve the exploration component of the optimization process when it is blocked in a local optimal.

3.1.5. Evolutionary Algorithms Properties

Diversity

The individuals of the population are spread all along the search space according to their encoded solutions. The performance of Evolutionary Algorithms (EA) is related to how representative of the search space the population is (Burke, et al., 2004). This is usually related to diversity. The diversity represents the structural or behavioral variety of a population. More diverse populations are usually related to representative samples while less diverse population are less representative. Since new individuals are usually generated based on previous ones' characteristics, having a diverse population is essential for the optimization process. Therefore, more diverse populations improve EC optimization process while poor diverse populations may produce premature convergence to local optima (Burke, et al., 2004).

Diverse initial population is crucial for evolutionary computation; however, it should not be forced all along evolutionary process since it may slow down convergence process (Burke, et al., 2004). Regular evolutionary operators affect diversity. Thus, they are useful to control population diversity. The population initialization is usually setup to provide a very diverse population that represent a wide range of possible solutions. The crossover mostly cause loss of diversity since it focusses the optimization process on promising solutions (Langdon, 1998). However, there are some crossover operators designed to provide tools to increase

the diversity. The mutation can be applied to increase diversity during evolutionary process.

There are several proposals to measure population diversity in GP (Burke, et al., 2004) (Hien & Hoai, 2006). These proposals can be classified in structural and behavioral diversity measurements. The structural diversity is mainly related to the genotype diversity while the behavioral diversity is more related to the fitness or the phenotype. Although these diversity measurements are defined as different paradigms, there is a strong correlation between the structural and the behavioral diversity. Since identical genotypes represents same behaviors, a loss in genotypic diversity will also produce a loss in behavior diversity. However, a rise on genotypic diversity not necessarily means a behavioral diversity rise. The structural diversity was firstly mentioned by Koza (Koza, 1992), who defined variety to measure the genotypic diversity. The variety measures the percentage of different genotypes in a population. Langdon argued that although this measurement provides poor information about population diversity, it is a sufficient upper bound of the population diversity (Langdon, 1998).

In GP and GGGP, two different genotypes may represent the same phenotype or behavior. This is mainly caused by the appearance of introns in the individual's genotype and the search space ambiguity. Introns do not affect individual's fitness. If introns are abundant in the population, the genotypic diversity may show incorrect measurements that are not correlated to the real population diversity. The ambiguity may also produce distorted diversity measurements since two completely different individuals may represent the same behavior, even though these individuals do not have any introns. To avoid diversity distortions when introns are present or the search space is ambiguous, the behavioral diversity is presented as a better and more real diversity measurement (Hien & Hoai, 2006). However, the behavioral diversity is usually more complex to obtain since it usually depends on the problem to solve and it is not either accurate.

Although there are several structural and behavioral diversity measurements that provide more detailed information about population diversity, Koza's variety is

widely used due to its simplicity. However, it must be considered that Koza's variety only provide information about the percentage of different individuals and not about how representative of the search space the population is. Moreover, most of evolutionary algorithm avoid having duplicate individuals so Koza's variety does not provide any diversity information.

Locality

Variations in the genotype produces also variations in the phenotype. The way phenotype varies according to the genotype variations is crucial for EAs performance (Vanneschi, et al., 2014). When small changes in the genotype produces small changes in the phenotype, it is said that locality property is complied (Galvan-Lopez, 2009). It has been proved that maintaining diversity is not sufficient for preserving the good performance of EAs but also it is necessary to ensure the locality (Galvan, et al., 2013) (Galván-López, et al., 2011) (Gottlieb & Raidl, 2000) (Uy, et al., 2013) (Hoai, et al., 2006). However, keeping locality from genotype to phenotype is usually intricate since it is directly related to the encoding systems. There are two main approaches to seek keeping locality by means of the variation operators: the syntactic (genotypic) and the semantic (phenotypic) approach (Uy, et al., 2013), (Uy, et al., 2010) (Hoai, et al., 2006). The syntactic locality seeks the variation operators produces small changes on the genotype, while the semantic approach focuses on producing small changes in the phenotype. On the one hand, the syntactic locality reduces the bloat and slightly increase the ability of GP to generalize. On the other hand, the semantic locality, in addition to reduce the bloat, it significantly improves the ability to generalize and the EAs performance (Uy, et al., 2010).

3.2. Genetic Algorithms

Genetic algorithms are one of the most popular EAs. A genetic algorithm is a metaheuristic optimization technique that borrow ideas from natural evolution (Darwin, 1859) to perform an evolutionary process to vectors of fixed size (Holland, 1992) (Goldberg, 1989) (Goldberg & Holland, 1988). Solutions are usually encoded

in vectors G (genotype) of fixed size l, which comprise a set of elements $g_1, g_2, \dots, g_l$ (genes), which values (alleles) belongs to an alphabet A, $g_k \in A$ for each $k \in \{1,2,\dots,l\}$. According to the alphabet's cardinality $|A|$, they can be classified as finite or infinite alphabets. The finite alphabet approaches are usually related to binary codifications such that $\forall g_k \in A,\ g_k \in \mathbb{B}, \mathbb{B} = \{0,1\}$, while the infinite alphabet approaches are related to real numbered codifications, $\forall g_k \in A, g_k \in \mathbb{R}$ or $\forall g_k \in A, g_k \in \mathbb{N}$. On the one hand, binary vectors can be employed to represent any kind of solution. However, the AG performance strongly depends on the search space codification. On the other hand, real numbered vectors are more common in mathematical optimization problems. Thus, the search space bias is less frequent. Figure 3 shows an example of binary and real numbered vectors of size 6.

Figure 3: Binary and real numbered vector of size 6

The operators used in the evolutionary process are selection, variation and replacement. Two variation operators are commonly applied: the crossover and the mutation. According to the vector codification, variation operators proceed in different ways. Finite alphabet crossovers usually apply vector recombination while infinite alphabet approaches use mathematical methods to modify vectors. Figure 4 shows some crossover examples for finite alphabet codification (Gwiazda, 2006) (Font, et al., 2009). For each crossover example, the pair of vectors on the left represents the parents, while the pair of vectors on the right the offspring. The arrow containing the symbol ×, represents the crossover direction. The single-point crossover randomly chooses an index $i \in \{1,2,\dots,l\}$ of parents' vector. Then, the offspring is generated by combining the vectors obtained after splitting parent vectors at i position. That is, given two parents vector $X = x_1, x_2, \cdots, x_l$ and $Y = y_1, y_2, \cdots, y_l$, such that $x_j, y_j \in \mathbb{B}$ with $j \in \{1,2,\cdots,l\}$, the resulting offspring will be the concatenation of the vectors $x_1, x_2, \cdots, x_i$ and $y_{i+1}, y_{i+2}, \cdots, y_l$, and the concatenation of the vectors $y_1, y_2, \cdots, y_i$ and $x_{i+1}, x_{i+2}, \cdots, x_l$. The N-point crossover, or multi-point crossover performs the same operation but providing N

different splitting indexes. Figure 4 shows an example with $N = 2$. Finally, the uniform crossover provides a mask vector that determines when the genes must be recombined, so that, a 1 at i mask vector position determines that the genes at position i are exchanged in the offspring, while a 0 determines not to be exchanged.

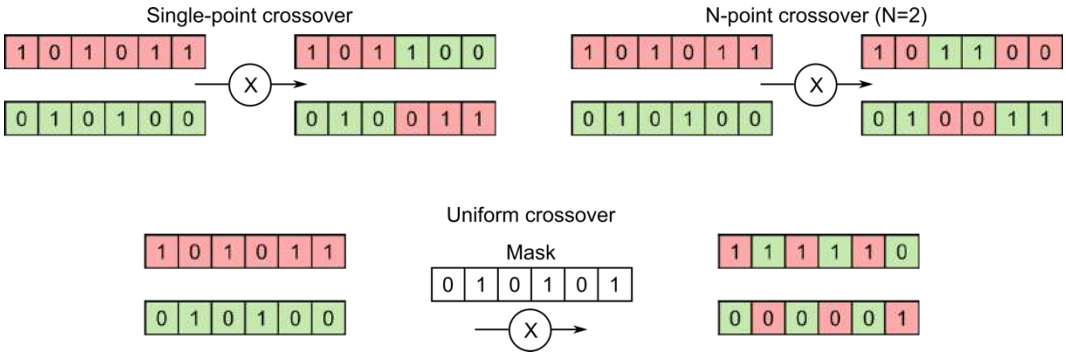

Figure 4: Crossover examples for finite alphabet genetic algorithms

Mathematical crossover methods can be also applied for finite alphabet GAs. The generalized crossover (Barrios Rolanía, et al., 1992) (Font, et al., 2009) uses binary-integer codification to provide a new search space where the evolutionary process takes place. Let $\mathbb{B}^l = \{0,1\}^l$ be the set of all possible binary vectors of length l and $b: \mathbb{B}^l \to \mathbb{N} = \left\{x \in \mathbb{B}: b(x) \in \mathbb{N} | \, l \in \mathbb{N}, b(x) \in \{0, 1, \cdots, 2^l - 1\}\right\}$ a bijective function that maps binary vectors of length l with natural numbers. Given two genotypes $X, Y \in \mathbb{B}^l$, the offspring $X', Y' \in \mathbb{B}^l$ is obtained according to $X' \in b^{-1}([b(X \wedge Y), b(X \vee Y)])$ and $Y' = b^{-1}\big(b(X) + b(Y) - b(X')\big)$ such that $X', Y' \in b^{-1}([b(X \wedge Y), b(X \vee Y)])$ where $\wedge$ and $\vee$ are the logic *and* and *or* operators.

Infinite alphabet crossovers usually apply mathematical operations to obtain the new offspring. Some of the most common crossover operators for real codification are presented (Gwiazda, 2006) (Palma Méndez & Morales, 2008) (Font, et al., 2009). In general terms, given two vectors $X = x_1, x_2, \cdots, x_l$ and $Y = y_1, y_2, \cdots, y_l$ such that $x_j, y_j \in \mathbb{R}$ with $j \in \{1,2, \cdots, l\}$, the crossover operators produce n new vectors $O^k = o_1^k, o_2^k, \cdots, o_l^k$ such that $o_j^k \in \mathbb{R}$ with $k \in \{1,2, \cdots, n\}$, $j \in \{1,2, \cdots, l\}$. The arithmetic crossover (Michalewicz, 1996) produce two new vectors such that $o_j^1 = rx_j + (1-r)y_j, o_j^2 = ry_j + (1-r)x_j$ with r a random number such that $r \in [0,1]$. Flat crossover (Radcliffe, 1991) produces n random vectors such that $o_j^k \in$

$[\min\{x_j, y_j\}, \max\{x_j, y_j\}]$. Linear crossover (Wright, 1991) produces three new vectors ($n = 3$) such that $o_j^1 = \frac{x_j + y_j}{2}$, $o_j^2 = \frac{x_j + y_j}{2} + |x_j - y_j|$, $o_j^3 = \frac{x_j + y_j}{2} - |x_j - y_j|$. Blend Crossover (BLX-$\alpha$) (Eshelman & Schaffer, 1992) produces n random vectors such that $o_j^k \in [\min\{x_j, y_j\} - I\alpha, \max\{x_j, y_j\} + I\alpha]$ with $\alpha \in [0,1]$ and $I = |x_j - y_j|$.

Mutation operator randomly modifies vector values so that new search space areas can be explored. Given a vector value $x \in A$, the mutation operator assignas a new value x' such that $x' \in A,\ x' \neq x$. When using binary alphabets, the mutation operator commonly assigns a 1 to x when $x = 0$ and 0 when $x = 1$. This mutation operator can be extended for real codification alphabet so that the x' is not uniformly obtained from $\mathbb{R}$, but it is chosen from a finite interval centered in x (Palma Méndez & Morales, 2008).

3.3. Genetic Programming

Genetic programming (GP) is a technique of evolutionary computation based on genetic algorithms (GA) designed to optimize programs (Koza, 1992) (Langdon & Poli, 2013). Genetic algorithms are widely applied due to the simplicity-efficacy trade-off for a large variety of problems. However, GAs are not able to obtain solutions of different length to the initially specified. GP is mainly design to deal with this limitation. Instead of evolving vectors of fixed size, GP evolves programs of variable size. Programs are encoded on syntax trees (Langdon & Poli, 2013). Figure 5 shows an example of syntax tree of the program $(x \cdot y) + \cos(\pi)$. A syntax tree represents the abstract syntactic structure of a program. Each node represents a symbol of the program and the edges between them, the way they are structured.

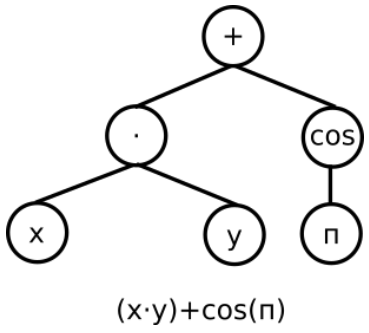

Figure 5: Genetic programming syntax tree example

There are two types of nodes: inner nodes and leaf nodes. Leaf nodes, denoted as terminals, are the variables and constants of a program such as x, y and π of Figure 5. Inner nodes, denoted as functions, are any kind of operators applied to the terminal symbols such as the sum $(+)$, the cosine $(\cos)$ and the product $(\cdot)$ of Figure 5. The superset of both, the terminals and function symbol sets, is denoted as the primitive set. The primitive set defines the set of all possible symbols that can be used in programs.

GP borrows general optimization process specification from GAs (Langdon & Poli, 2013). The GP evolutionary process follows Figure 2 flow chart. Every GP generation some of the best-fit individuals are selected, crossbred, its offspring mutated, and then some of the worst adapted individuals are replaced by the new individuals. Firstly, the algorithm initializes the population. This is usually performed by randomly creating a set of new individuals that comply with the problem restrictions. Sometimes, solutions obtained in previous optimizations are recycle to set part of the initial population. The general procedure for syntax tree initialization starts setting a symbol from the primitive set as the syntax tree root. Then, if the generated symbol is a function symbol, new symbols are selected from the primitive set to assign each function's arguments, and so on (Langdon & Poli, 2013). Figure 6 shows an example of syntax tree initialization. The syntax tree initialization steps are shown from left to right and top to bottom.

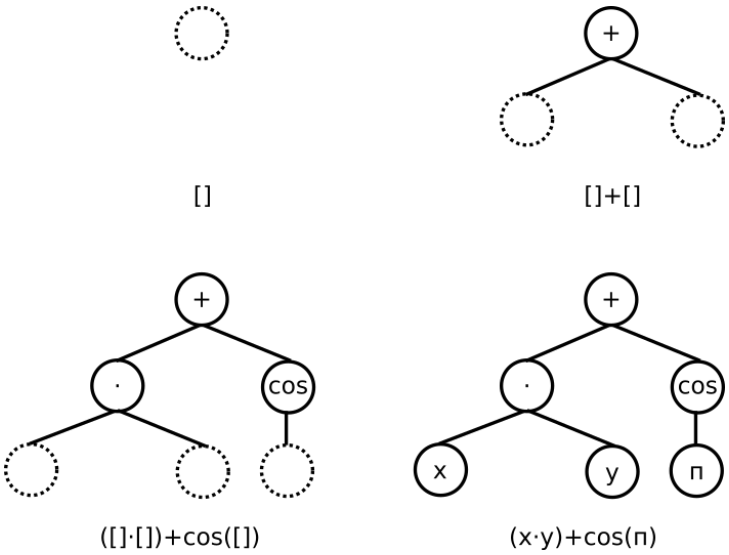

Figure 6: Syntax tree initialization example

Then, the evolutionary process starts. Every generation of the evolution, a selection operator is used to select some of the best individuals of the population. These individuals are the progenitors of the new offspring. The genotypes of these individuals are recombined by means of a crossover operator. This operation reproduces the inheritance process. The crossover is executed following a probabilistic distribution that specifies whether it is applied. Then, an offspring containing the main characteristics of their progenitors is obtained. Most of GP approaches apply crossover to two syntax trees. The crossover is performed by exchanging two subtrees of parents' syntax trees. Figure 7 shows an example of GP crossover. The two upper syntax trees represent the parent while the two lower ones, the offspring. The arrows containing the symbol ×, represent the crossover direction. Two crossover points are selected by randomly choosing a node of each parent's syntax tree. For Figure 7, the nodes y and sin of parents' syntax trees represent the crossover nodes. Each subtree node is released of its corresponding syntax tree and replaced by the other subtree producing the offspring.

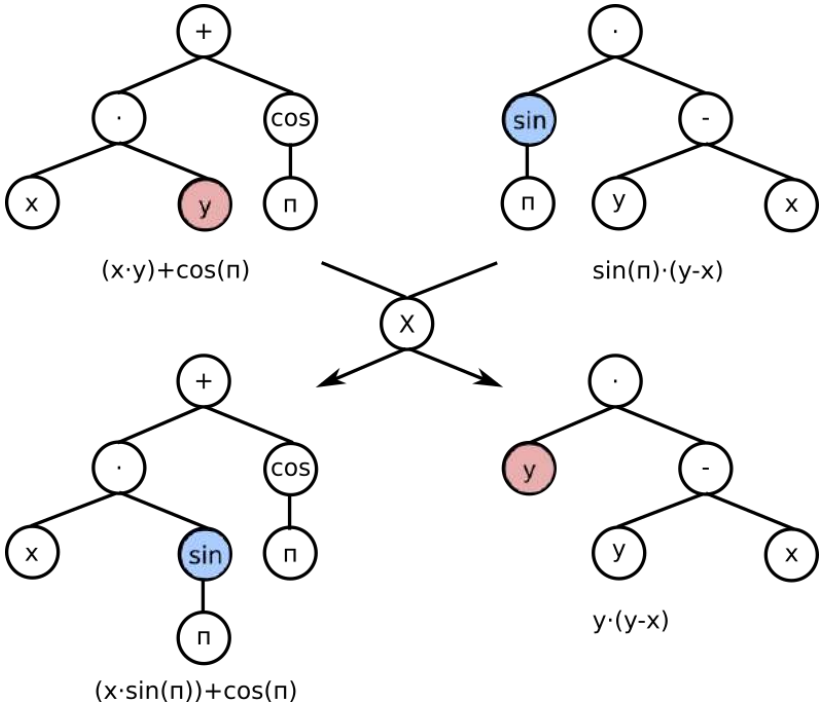

Figure 7: Genetic programming crossover example

Right after, the mutation is applied following a probabilistic distribution. The mutation produces a random modification on the offspring genotype. It randomly

selects a node of a syntax tree and generates new subtree. Then, the replacement operator is executed and the resulted individuals are inserted into the population after some of the worst individuals of the populations are deleted.

3.3.1. Properties

Closure Property

In GP, function set is commonly required to comply with the closure property (Koza, 1992) (Poli, et al., 2008) (Koza, et al., 2006). The closure property ensures the generation of valid solutions throughout all evolutionary process. It can be achieved by ensuring type consistency and evaluation safety (Poli, et al., 2008). The type consistency ensures crossover recombination takes place avoiding invalid syntax programs on the resulting syntax trees. This is possible, for example, by guarantying every primitive set symbol have the same type. That way, any subtree can be located in any function argument of the function set and produce new valid programs. However, this approach is very restrictive and reduce GP usefulness. A less restrictive alternative is to extend GP system (Montana, 1995). Instead of forcing the same type for every symbol of the primitive set, GP can check if a subtree or symbol match the function argument type where it is set. The evaluation safety is required to avoid errors during the runtime evaluation of the programs such as dividing by cero a number. These errors are not easily detected and they are revealed during program evaluation when an error is produced. A common solution to this issue is applied protected versions of functions that may cause evaluation errors, so that, the function returns a predefined value when the function detects the evaluation would fail (Poli, et al., 2008).

Sufficiency

The sufficiency determines the capability to obtain a solution for the problem to solve by only using the elements of the primitive set (Poli, et al., 2008). The sufficiency property is hardly guaranteed since there is no procedure able to check if the symbols in the primitive set are sufficient to solve a specific problem. Only

when there is some previous knowledge of the problem to solve, determining the sufficiency of the primitive set may be possible.

Bloat

A common known issue in most of GP approaches and other variable-length optimization techniques is called bloat: an excessive code growth without a corresponding improvement in fitness (dal Piccol Sotto & de Melo, 2016). The bloat is usually related to the appearance of introns: fragments of code which do not have any effect to the fitness (Castelli, et al., 2014). The code bloat may lead to several problems such as memory depletion, slow evaluation of large programs, slow convergence since the modification of introns does not produce any improvement and obtaining large final solutions with several introns. Thus, this is a phenomenon to avoid during GP evolution.

There are several theories about the origin of bloat (Poli, et al., 2008). The oldest ones argue that the problem may be cause either by the inability to accurately replicate the parents' functionality (McPhee & Miller, 1995), by the introduction of introns on genotype (Soule & Foster, 1998) or by the fact that there are more larger syntax trees than shorter ones with the same fitness (Langdon & Poli, 1998) (Langdon, et al., 1999). However, recent theories claim there is no strong evidence to assert that main cause of bloat is caused by the crossover operations or the appearance of introns (Trujillo, et al., 2016). Indeed, it is shown that crossover do not bias the evolutionary process to larger syntax tree, but instead, it leads the population to a Lagrange distribution of second kind where the average size of trees is not affected and small programs have higher frequency than larger ones (Poli, et al., 2007) (Dignum & Poli, 2007). Most of recent theories agree to point out the selection operation as the main cause of bloat (Silva & Costa, 2009). However, the most accepted theory nowadays, the crossover bias theory, states that the bloat is not only cause by the selection operation but also by the crossover bias towards small syntax trees (Poli, et al., 2007) (Dignum & Poli, 2007). Since very small program have no chance of solving most of problems, the programs above the average size are likely selected, and therefore, the population average size increase.

3.3.2. Population Initialization Variants

GP starts generating the initial population. The initial population is generated randomly to obtain a uniformly distributed sample of the search space. This process is usually performed by randomly choosing symbols from the primitive set. However, random not always means uniform and initialization process must be biased to artificially achieve uniform populations. Additionally, there are some constrains to consider during initialization. The closure property is one of them (Mckay, et al., 2010). Individuals generated during population initialization should comply with closure property, that is, ensure the type consistency and the evaluation safety. That way the population individuals will always be valid solutions. Bloat is another common problem in GP to consider during population initialization (dal Piccol Sotto & de Melo, 2016). Size bounds has been postulated as a simple solution to limit code bloat in GP (Poli, et al., 2008).

The full and grow methods are early population generation algorithms (Poli, et al., 2008) for GP. Both control the maximum depth that syntax trees can reach. The depth of a node is defined as the distance, in number of arcs, to reach that node starting from the root node. The tree depth is the largest depth of the tree's nodes. Both methods take as arguments the maximum depth of syntax trees and the primitive set (terminal and function sets). The full method is designed to build syntax trees that always reach the maximum defined depth in every branch of the tree. To do so, it iteratively and randomly chooses nodes from the function set until the specified maximum depth is reached. Then terminal nodes are chosen to finish the branch. This method usually produces dimensionally alike syntax trees, as their branches always have the same depth. The dimension only differs on breadth, which mainly depends on the number of arguments of the function set elements.

The grow method is presented as an alternative that build syntax trees with different branch depths. Instead of iteratively choosing nodes from the function set, the grow method select them from the primitive set. This way, the branch depth increases when a function symbol is selected from the primitive set, and it stops

growing when a terminal symbol is selected. Similarly, to the full method, if the maximum depth is reached, only nodes from the terminal set can be selected. Unlike the full initialization strategy, the grow method produces more dimensionally diverse syntax trees in both breadth and depth dimensions. However, neither the full nor the grow methods generate diverse syntax trees in size and shape.

The ramped half & half algorithm combines both: half of the population is generated using the full method and the other half using grow (Koza, 1992). Additionally, instead of using a static maximum depth for the whole population, it uses a range of depth bounds that improves size diversity (ramped strategy). Even though half & half improvements seek more dimensionally diverse trees, they are still far from optimal distributions (Poli, et al., 2008). The size and shape probability distributions are difficult to control since they strongly depend on the initial setup of the primitive set. Since the nodes are randomly selected from the primitive set, the proportion of function symbols compare to terminal symbols is determining since it can produce strong biases on the initial population. The higher is the function set cardinality compared to the terminal set, the bigger will be the syntax trees size. Primitive sets with a higher function set cardinality will lead the half & half and grow methods to behave as the full initialization method. However, if the terminal set cardinality is higher than the function set cardinality, the generated syntax trees are usually smaller. Additionally, the number of arguments of function symbols may also produce bias on initial population since they vary the number of possible branches per node. The more branches a node have, the bigger the syntax tree will be. Again, the proportion of functions with different number of arguments will shape the population to syntax trees of different sizes. The ramped half & half is criticized and praised for building small and bushy trees (Poli, et al., 2008). Bushy trees are usually good at solving symmetric problems, such as parity. However, this kind of trees does not fit for many other problems, such as Santa Fe Trail problem; hence GP optimization process may not find a proper solution (Langdon & Poli, 1998).

Several methods have been developed claiming to provide better uniform initial populations (Luke & Panait, 2001). These initializations generate more asymmetric trees where the leaf nodes depths strongly vary compared to the ramped half and half method. These methods improve GP performance for problems where syntax tree solutions are asymmetric. However, the ramped half and half still outscore them when dealing with symmetric problems. In order to obtain a more uniform population, two approaches based on a bijective algorithm (Poli, et al., 2008) were presented (Böhm & Geyer-Schulz, 1996) (Iba, 1996). Both algorithms gather information of syntax tree search space to produce a probability model for choosing those nodes and tree sizes that uniformly initialize syntax trees. Since this process is computationally expensive, it is executed before initialization starts. Then, the syntax tree generation is performed by recursively choosing the nodes and tree sizes following the probabilistic model. The recursive function starts choosing an initial node. Then, for every branch of that node, a tree size is assigned and the recursive function is called again. This process continues until the maximum size of the branches is reached.

The ramped uniform (Langdon, 2000) is a variant closer to the gradualist approach ramped half & half. This method allows user to provide a range of desirable tree sizes. It firstly precomputes the different possible combinations of functions and it classifies them according to the resulting tree sizes. Then, a probabilistic model for the functions combinations is created according to the number of trees of each size. Finally, syntax trees are generated by randomly choosing a precomputed functions combination of a given random size. These three last approaches are criticized for their computational cost at gathering syntax trees information. Although it is possible to reduce the cost by applying dynamic programming, the computational cost is still polynomial (Luke & Panait, 2001).

Other approaches, like PTC1 and PTC2, were designed to initialize syntax trees according to some user structure requirements (Luke, 2000). These methods strongly reduce the computational cost. However, the structure uniformity of syntax trees is reduced in favor of complying with user requirements. The PTC1

method is a variant of the grow method where the produced syntax trees sizes are in average close to a user-defined expected tree size. The core of this method is a precomputed probability model that specifies whether to choose a function or a terminal node. The probability model guides the syntax tree initialization to tree sizes close to the expected. The model is obtained by only using the function set cardinality and the number of variables of each function. Since, the probability only depends on known data, it can be precomputed before the initialization starts. Additionally, the user can define the probability to choose a terminal or a function symbol within their respective sets providing more control over the symbols selection. The author claims that this initialization method is very simple and computationally efficient. However, sometimes it is not suitable since it does not provide control over the variance in syntax tree size and it usually produces several small trees. The PTC2 is an alternative to PTC1 that require trees to be of a given size or slightly larger. Unlike the PTC1, the PTC2 lets user define a probabilistic model of desirable tree sizes. Every time the algorithm generates a syntax tree, it randomly chooses a size from the user-defined tree sizes model. Then, it generates a tree by randomly extending it with function symbols until the selected tree size is reached. Finally, the pending function variables are completed with terminal symbols. Although this solution increases user control, it does not actually ensure tree sizes distribution.

The Random Branch approach (Chellapilla, 1997) provides better control on initial syntax tree sizes. This approach guarantees producing trees of a given size or slightly smaller. The initialization recursively chooses function symbols whose number of arguments do not exceed a maximum bound size. If there is no function that complies with the argument number restriction, a terminal symbol is selected. Every time a function symbol is selected, the maximum bound size is evenly distributed within the arguments of the function so that, the maximum bound for next function is obtained by dividing the parent bound by the number of arguments of the parent function. The maximum number of arguments for the root is set to the maximum tree size. The GP performance may be substantially decreased when

using this approach since it is quite common that part of search space syntax trees cannot be generated by this method (García-Arnau, et al., 2007).

3.4. Grammar Guided Genetic Programming

Genetic Programming has been postulated as a powerful and simple optimization algorithm when dealing with variable size solution problems. However, keeping the closure property throughout the whole evolutionary process while the performance is not affected is still a challenging open issue (O'Neill, et al., 2010) (Mckay, et al., 2010). There exist ways to keep closure property on GP's function set. A well-structured function set is the ideal setup for GP, but most of the times this is not feasible. Sometimes the closure property can be forced by using, for example, a conversion mechanism between types for type consistency or by using protected version of functions for evaluation safety. However, these mechanisms usually introduce bias on the optimization process that decrease GP performance. Extending GP by adding some constraints has been proposed as an effective solution (Poli, et al., 2008).

Type constraining can be applied to extend GP and preserve the closure property. Dimensionally aware GP (Keijzer & Babovic, 1999) is an approach of type constraining designed to work in scientific environments where multiple measurement units are used. This extension labels problem expressions to provide dimensionality information. These labels are used to preserve the closure property by matching expressions typing and dimensionality. Moreover, a dimension transformation function is provided to securely resolve dimensional violations that may occur during the GP crossover. However, fitness must be biased to reduce the use of this function since it might introduce non-physically meaningful expressions. Strongly Typed GP (Montana, 1995) (Haynes, et al., 1996) approaches also implements type constraining. These approaches assign a type label to every terminal, function argument and function return value. The population initialization, crossover and mutation are modified to only assign terminals and

functions that match the type label. However, these approaches do not ensure the dimensional consistency.

Syntactic (Gruau, 1996) and semantic (Ványi & Zvada, 2003) constraints can be used delimit search space boundaries. Formal grammars have been used (Horner, 1996) in a strongly typed approach of GP that effectively preserve type constraining during the crossover and mutation. Nevertheless, this approach may produce invalid derivation tree during population initialization (Ryan, et al., 1998). A dimensionally constrained formal grammar has been proposed too for GP (Ratle & Sebag, 2000). This approach include grammar rules with dimensionality constrains that preserves closure property throughout all evolutionary process. CFGs have been also postulated as an effective solution to comply with the closure property throughout all evolutionary process (Whigham, 1995). This approach, called grammar guided genetic programming (GGGP), encodes the solutions as CFG derivation trees. The leaves of the derivation tree represent the encoded solution, while the derivation tree, how the solution is build using the CFG. The CFG delimit the problem search space so that every derivation tree generated by the CFG encodes a feasible solution. In general terms, GGGP follows the same evolutionary process as GP, excepting for the population initialization, crossover and mutation operators. In GGGP, these operators use a CFG so that the resulting individuals are always feasible solutions belonging to the search space of the problem. The general procedure for the initialization of derivation trees is described in Table 6 (García-Arnau, et al., 2007). Starting from the axiom of the grammar and continuing with the produced nonterminal symbols, productions rules are iteratively applied until only terminals are produced. Given the axiom or any nonterminal symbol, production rules are randomly chosen from the set of production rules with the same LHS nonterminal symbol.

Table 6: GGGP population initialization algorithm.

Repeat N times:

1) *Set axiom as current nonterminal symbol A*
2) *Randomly choose a production rule* $r: A ::= \gamma \in R_A$
3) *For every nonterminal symbol* $B_i \in \gamma$
   a) *Repeat from step 2 with* $A = B_i$

If the CFG is non-stochastic, given a non-terminal A, productions are randomly selected applying a uniform distribution over the set R_A. If the CFG is stochastic, production rules are randomly selected using the probability model defined by the set of $P(r)$ for each R_A. The GGGP crossover operation only adds a restriction to GP crossover: the crossover nodes must contain the same symbol (Whigham, 1995). Figure 8 depicts an example of a crossover operation in GGGP for two derivation trees of the grammar $G_{Exp} = (\{E\}, \{1, +, *\}, R, E)$ where $R = \{E ::= E + E,\ E ::= E * E\ \ E ::= 1\}$.

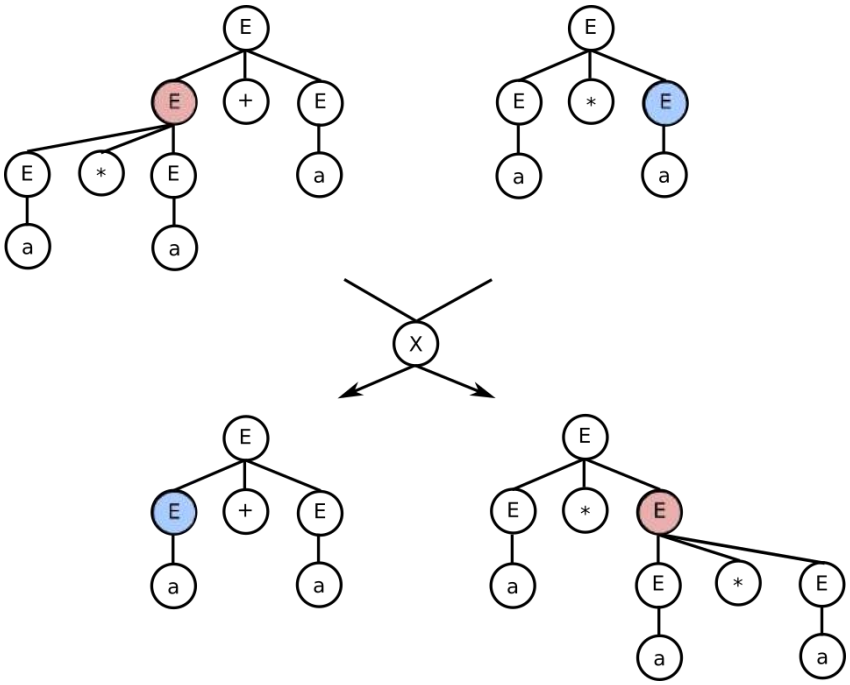

Figure 8: GGGP Crossover example

The two upper derivation trees represent the parent while the two lower ones, the offspring. In this case, the parents encode the solutions $(a * a) + a$ and $a * a$, and the resulting offspring encodes $a + a$ and $(a * a) * a$. The arrows containing the symbol ×, represent the crossover direction. Two crossover points are selected by randomly choosing a node of each parent's derivation provided that the crossover nodes must contain the same symbol. In this case, the nonterminal symbol E. Each subtree node is released of its corresponding derivation tree and replaced by the

other subtree producing the offspring. Note that this operation is performed to a copy of the progenitors' derivation trees so that progenitors' trees are not altered.

3.4.1. Population Initialization Variants

The GGGP population initialization research has focused on improving the evolutionary process in two different ways: improving distribution uniformity by directing initialization to some specific sizes or characteristics, and reducing code bloat by delimiting individuals size. GP already showed the importance of defining bounds on population initialization to avoid code bloat. These practices have been inherited from GP and the maximum depth bound was included in the very beginning of GGGP (Whigham, 1995) (Gruau, 1996) (Whigham, 1995). Bounds benefit the evolutionary process by limiting code-bloat and forcing crossover to perform small changes on genome. Maximum derivation tree depth bounds have been postulated for recursive CFG (Gruau, 1996), and both, nonrecursive and recursive CFG (Whigham, 1995) (Whigham, 1995). The maximum depth bound is defined by user while the minimum one is set to the minimum depth of axiom derivation trees rewriting.

Seeking uniformity in GGGP population initialization is not easy as CFGs contain inherent bias that shape population initialization in different ways depending on the grammar definition (Poli, et al., 2008) (Yu, et al., 2006). Small changes in problem search space definition may produce unintended biases (Poli, et al., 2008). Moreover, CFG provides some syntactical constraints that also affect the evolutionary process, e.g. promoting the generation of some language sentences over others (Yu, et al., 2006) (Whigham, 1995). Initialization techniques developed for GP have been adapted to be used in GGGP and improve uniformity (Whigham, 1995) (Whigham, 1995). In particular, the ramped approach is applied to randomly generate derivation trees ranging from the CFG minimum depth to a defined maximum depth. However, since GGGP initialization is guided by the CFG, many of the individuals can be rejected when production rules that surpass the depth bound are selected. Accordingly, the population would be finally condensed in some

specific derivation trees that use production rules with low probability to surpass depth bound during initialization. To avoid selecting production rules that violate the depth bounds, rules are labeled with the minimum depth of their derivation trees rewriting (Whigham, 1995) (Whigham, 1995) (Gruau, 1996). During the derivation tree generation, only the production rules which minimum depth plus the current nonterminal symbol depth do not exceed the maximum depth bound can be applied. The dimensionally constrained GGGP system (Ratle & Sebag, 2000) also introduces a very similar initialization process that ensures the generation of derivation trees below a depth bound. Not only production rules are labeled with the minimum depth of their derivation tree rewriting, but also the nonterminals. The lowest minimum depth over all the associated production rules is assigned to the nonterminal symbols. These approaches are later extended and formalized (García-Arnau, et al., 2007). Table 7 defines the recursive algorithm to obtain the minimum depth, denoted as $L()$, of all production rules and symbols of a CFG.

Table 7: Production rules and symbols minimum depth definition procedure.

1) *Set every terminal symbol minimum depth to 0.*
2) *Set axiom as current nonterminal symbol* A
3) *For each production rule* $r: A ::= \gamma \in R_A$.
   a) *For each nonterminal symbol* $B_i \in \gamma$
      i) *Repeat from step 3 with* $A = B_i$.
   b) *Set* r *minimum depth as 1 plus the highest minimum depth within all* $C \in \gamma$ *such that* $C \in \{\Sigma, V\}$.
4) *Set* A *minimum depth as the lowest minimum depth of all* $r \in R_A$.

The minimum depth of a CFG is determined by the minimum depth of the CFG's axiom. Table 8 defines the regular GGGP population initialization algorithm with depth bounds to produce N derivation trees with a maximum depth bound D. This algorithm requires the minimum depth of all CFG's productions and symbols so algorithm of Table 7 must be firstly executed.

Table 8: GGGP population initialization algorithm with depth bounds.

Repeat N times:

1) *Set maximum depth bound d to a random value between the CFG minimum depth and D.*
2) *Set axiom as current nonterminal symbol A and current depth $CD = 0$*
3) *Randomly choose a production rule $r: A ::= \gamma \in R_A$ such that it complies with $CD + L(r: A ::= \gamma) \leq d$.*
4) *For every nonterminal symbol $B_i \in \gamma$*
   a) *Repeat from step 3 with $A = B_i$ and $CD = CD + 1$*

Breath bounds have also proven its benefits in GGGP (Ványi & Zvada, 2003), where every nonterminal symbol and production rule is labeled with the minimum breadth of the set of sentences they generate.

There are some other research works that have improved the evolutionary process by biasing the population to some specific characteristics. The population initialization distribution can be shaped by increasing the probability of generating some promising solutions. Stochastic CFGs have been applied to modify the population initialization distribution (Ratle & Sebag, 2001) (Ratle & Sebag, 2002) (Tanev, 2004). During the population initialization, the weights assigned to each production rule determine the probability of choosing one of them against the rest of production rules with the same LHS nonterminal symbol.

3.5. Genetic Programming Variants

The search space defined by CFGs is usually complex. In addition to GP tree consistency, GGGP operators must comply with CFG constraints. Moreover, evolutionary operators must be parametrized to comply with other constrains like maximum derivation tree depth (Ványi & Zvada, 2003). This complexity has limited the appearance of successful methods for optimizing programs. Effective optimization algorithms such as simulated annealing, hill climbing, random search have been applied to GGGP (O'Sullivan & Ryan, 2002). Differential evolution (O'Neill & Brabazon, 2006) and particle swarm algorithms (O'Neill & Brabazon, 2006) have been also applied. In general terms, all of them have fail improving the

base performance. This happens because highly exploitative methods are not suitable for CFG search space compare to other methods such GGGP optimization where the crossover exploitation-exploration trade-off is more optimal (Mckay, et al., 2010). However, the GGGP research has been focused on creating new optimization paradigms for CFGs since the GGGP variation operators do not offer many improvements possibilities (Mckay, et al., 2010). These new optimization paradigms seek keeping a suitable explotaition-exporation trade-off while trying to improve GP desirable properties such as building block preservation, locality, bloat avoidance, for which GGGP has shown some issues. Sufficiency has been target too in approaches where the grammar is not able to find a feasible solution.

In GGGP, the evolution solely relies on tree mutation and crossover. Due to CFG constraints, these operations do not provide any control to perform local search (Mckay, et al., 2010). On the one hand, given two similar derivation trees, a crossover operation may produce completely different derivation trees. On the other hand, the result of a mutation operation strongly varies depending on the mutated derivation tree node, so that, the mutation of nodes close to derivation tree leaves usually produce small variations while the mutation of nodes closer to the derivation tree root, produce major variations. Moreover, variations on the genotype may lead to unrelated variations on phenotype, so that a small variation in genotype may produce a larger variation on phenotype. The preservation of building blocks is another desirable feature of GP for which the genetic operators have not achieved remarkable results. Although, the building block hypothesis can be considered a good approximation to GP behavior with one-point crossover (Poli, 2000), the genetic variation operators preform subtree changes to random derivation tree nodes that lead to highly destructive effects (O'Reilly & Oppacher, 1994). The code bloat is another issue of genetic evolutionary algorithms. The crossover bias theory states that the bloat is related to the crossover bias. This bias leads the population to a Lagrange distribution of second kind where very small programs with low probability to solve the problem are more frequent than larger

ones (Poli, et al., 2007) (Dignum & Poli, 2007). In the following subsections, some of the most popular GP variants developed to solve these issues are presented.

3.5.1. Grammatical Evolution

Grammatical evolution (GE) is a linearization of GGGP that replaces derivation trees by integer strings that encode the position of the production rules required to obtain a program (O'Neil & Ryan, 2003). This approach provides new possibilities to improve variation operators since changes in genotype are easily controlled. Moreover, since programs are linearly represented, the research work developed for GAs can be recovered for GE. This approach reduces the code bloat issue compared to tree-based GP systems. However, the linearization of the CFG search space aggravates the locality issue since small variation operations may produce large variations in the phenotype. In general, GE variation operators are more likely to produce large modifications in programs than GGGP. This produces a bias in the optimization process towards a more exploratory behavior (Mckay, et al., 2010).

3.5.2. Estimation of Distribution Algorithms for Genetic Programming

Estimation of distribution algorithm (EDA) (Larrañaga & Lozano, 2001) (Hauschild, 2011) is a variant of GAs that uses probabilistic models to drive the evolutionary process towards promising solutions. In general terms, EDAs has shown better performance than GAs. Moreover, there is large research background in statistical learning that help developing new satisfactory methods. In general terms EDAs algorithm work as follows: Every iteration, a probabilistic model describing the distribution of the current population characteristics is generated. The probabilistic model can be generated every generation or incrementally updated according to a learning rate that determines the weight of previous probabilistic models compare to the current population distribution (Hauschild, 2011) (Baluja, 1994). This model is later used to create a new population based on these characteristics. Instead of using genetic variation operators, EDA base the evolutionary process in the population initialization operator. Firstly, a random

population is generated. Then, every iteration a set of promising solutions is selected to create the probabilistic model. Then, the current population is replaced by a new population generated by a population initialization method that uses the probabilistic model to replicate promising characteristics of the progenitors.

This probabilistic approach is also applied for GP and GGGP (Shan, et al., 2006) (Mckay, et al., 2010) (Kim, 2013). Probabilistic model approaches are presented as a solution to reduce code bloat in GGGP, preserve building blocks and improve the evolutionary performance (Ratle & Sebag, 2002) (Mckay, et al., 2010). There are three major research lines determined by the way the population distribution is modeled: by means of CFG, trees or graphs (Kim, 2013). However, the research work is scarce in the graph modelling field (Boryczka, 2002) (Rojas, 2004), and is more related to Ant Colony Optimization methods (see Chapter 3.5.4).

Early GGGP approaches apply stochastic CFG to approximate the population distribution (Ratle & Sebag, 2001) (Ratle & Sebag, 2002) (Tanev, 2004). The stochastic CFG is gradually updated to create a probabilistic model that generate derivation trees using those production rules belonging to the promising solutions. The CFG production rule's weights are updated during the optimization according to individual's fitness. The production rule's weight appearing on well fitted individuals are increased while weights of poor fitted individuals are decreased (Ratle & Sebag, 2001) (Ratle & Sebag, 2002). However, the probability of selecting any production rule of a CFG or stochastic CFG not only depends on the probability of choosing that production rule within the set of production rules with the same LHS nonterminal symbol. Actually, the probability of choosing any production rule is determined by the probability of choosing that rule and the preceding's production rules (Mckay, et al., 2010).

More complex grammars have been applied to model the population distribution (Abbass, et al., 2002) (Shan, 2003). A stochastic parametric Lindenmayer system (L-system) (Prusinkiewicz, 2012) is applied to create a new probabilistic model approach (Shan, 2003). The L-system adds two new conditions to the LHS of the production rules: the depth and the location. These conditions indicate where the

production rule can be applied according to the depth of a node and its relative coordinate. The learning process involves two stages: probability learning and grammar structure refinement. The probability learning increases the probability of production rules which were applied to generate fitted individuals. The grammar structure refinement split production rules so that, there is one production rule for each different position of the LHS nodes. This way, the probability of applying a production rule not only depends on the rule itself but also on the position of the node in the tree.

Some EDA approaches try to address sufficiency issues by providing an optimization process that obtain more suitable grammars for those problems where the initial grammar is not able to find a feasible solution. To such aim, grammar learning systems are presented as an approach that optimize both: the stochastic CFG and the associated probabilities (Bosman & De Jong, 2004) (Hasegawa & Iba, 2009) (Shan, et al., 2004). These algorithms start with a basic stochastic CFG which is used to generate the initial population. Every iteration a new grammar is created according to the population characteristics. Then, a new population is sampled using the obtained grammar.

Probabilistic prototype trees (PPT) are used to represent the GP population distribution (Salustowicz & Schmidhuber, 1997). PPTs represent the set of all possible trees given a maximum tree depth. Each node of the PPT contains a probability vector indicating the probability of occurrence of the different primitive symbols. These vectors are updated every iteration according to the distribution of best population individuals. Then, when generating a new tree, symbols are selected following the node's probabilities vector.

A common characteristic of some of these methods is the model building. The model building technique is presented as solution to improve building block preservation and reutilization in probabilistic methods (Kim, et al., 2014). The building blocks and their relative and global location, called models, are identified. Then these models are reutilized when generating new programs. The models can be placed either in the original global position or in a new position where their relative

location is preserved. This approach benefits the evolutionary process in terms of performance. However, this improvement may not be worth it due to the complex tasks involved in the modularity process (Kim, et al., 2014).

3.5.3. Tree Adjoining Grammars

Tree adjoining grammars (TAG) are context sensitive grammars that combine elementary trees to produce the sentences of the language (Joshi & Schabes, 1997). The primitive elements of a TAG are elementary trees. There are two types of elementary trees: initial trees and auxiliary trees. The elementary trees comprise nonterminal and terminal symbols. All interior nodes of the elementary trees are labeled with nonterminal symbols. The leave nodes of the elementary trees are called frontier nodes. Initial trees must have at least one frontier node labeled with a terminal symbol. The frontier nodes of an initial tree labeled with nonterminal symbols are substitution nodes. The auxiliary trees are elementary trees that have at least one frontier node labelled with the same nonterminal symbol as the root node. This frontier node is called foot node. The operators used to produce the sentences of the language are the substitution and the adjoining. The elementary trees which root is labeled with the same nonterminal that the label of a substitution node can be attached to that node. This operation is called substitution. The adjoining operation can be applied to any node of an elementary tree labeled with the same nonterminal as the root and foot nodes of an auxiliary tree. The adjoining operation attaches the subtree of that node to the foot node of the auxiliary tree. Then, the auxiliary tree is attached to the original position of the node. TAGs have been also applied to solve GP problems (Hoai, et al., 2006) (Murphy, et al., 2012) (Abbass, et al., 2002). TAGs improve the performance of the GP evolutionary process since they do not have the inherit bias of CFGs (Murphy, et al., 2012). Moreover, phenotype-genotype transformation in TAGs are less disruptive. This fact facilitates the design of new variation operators or even the reutilization of GA methods (Mckay, et al., 2010).

3.5.4. Ant Optimization Variants

Ant colony optimization (ACO) is a probabilistic method for solving optimization problems whose search space can be expressed as a graph and the solution involves finding a path between two nodes (Dorigo, et al., 2006). This algorithm is bioinspired in the pheromone trails used by ants to mark the path to follow. Ants release pheromones as they wander looking for food. If another ant finds a pheromone trail, it might follow that path according to pheromone levels. The higher is the pheromone level, the higher the probability to follow that trail is. If the ant follows the trail, that trail will be reinforced as that ant releases more pheromones. Over time, the pheromones evaporate and the trail starts to disappear if ants stop going through that trail. This phenomenon can be used as an optimization method. Ants are asked to find a path between two nodes of a graph. As ants need less time to traverse shortest paths (better), they complete the travel and repeat it faster than longer paths (worse). Thus, the pheromone levels are higher in shorter paths than longer ones since ants traverse the shorter paths more frequently.

One of the research lines of estimation of distribution algorithms is focused on using stochastics graphs for modelling the population distribution (Boryczka, 2002) (Rojas, 2004). These approaches represent the GP search space as a graph. The probability to select the primitive symbols is defined by the probabilities associated to the stochastic graph. An ACO method is applied to optimize the probabilities of the stochastic graph so that it generates the solution for the problem at hand. ACO have been also applied to optimize the probabilistic model based on TAG approaches (Abbass, et al., 2002).

3.6. Hybrid Evolutionary Algorithms

Bioinspired in nature relationships and learning concepts, several approaches have been proposed to solve optimization problems where numerical terminal symbols are involved in GGGP. The symbiosis has been proposed to model a multi-level optimization approaches (Heywood & Lichodzijewski, 2010) (Migdalas, et al., 2013)

where two or more optimization techniques (e.g., GP and GA) hierarchically cooperate to find a solution for a problem at hand. Evolutionary symbiotic multi-level optimization can be mainly designed in two different ways: executing the optimization evolution processes consecutively or synchronously. Consecutive optimization firstly executes main optimization technique, like GP. Then, the obtained result is improved by locally applying another optimization technique, e.g. numeric optimization (Alonso, et al., 2009) (Panyaworayan & Wuetschner, 2002). In reverse order, local optimization can also be accomplished before main optimization technique starts (Cagnoni, et al., 2005). This approach presents an important drawback: since local variables are null, random or pre-optimized constants, they are not always suitable for all global solutions. Therefore, main optimization technique likely evolves to local optima. Synchronous optimization evolution tries to overcome this issue by alternatively executing different search and optimization techniques, e.g. numerical optimization and GP evolution. In this case, numeric optimization can be applied every GP iteration. Every time a new GP individual is created it is also optimized applying backpropagation (Couchet, et al., 2007), gradient descent (Topchy & Punch, 2001), linear regression (Keijzer, 2003), genetic algorithms (Cagnoni, et al., 2005), or differential evolution (Cerny, et al., 2008) (Mukherjee & Eppstein, 2012). Numerical optimization can also be applied after a certain number of GP iterations, or when GP is incapable to improve its fitness (Vanneschi, et al., 2006). However, synchronous approaches are computationally very expensive since they try to obtain the best terminals for every individual of the population.

There are also other approaches that attempts to overcome EC difficulties by means of learning techniques that seek Baldwin Effect benefits (Ackley & Littman, 1991) (Downing, 2001) (Hinton & Nowlan, 1987) (Whitley, et al., 1994). These approaches assume that learning some features during lifetime rather than awaiting them to appear in the genome is beneficial (Harley, 1981) because learning permits promising organisms to rapidly adapt themselves to their environment (Ackley & Littman, 1991). Beneficial learnt features can then be acquired in the genome.

Since, as a result, individuals' fitness improves, these individuals will more likely have an offspring and therefore these features will be inherited (Ackley & Littman, 1991) (Turney, et al., 2007). Lamarckian approach could facilitate the incorporation of learning knowledge in the genotype (Downing, 2001). However, it has been shown that the Baldwin strategy improves the results of Lamarckian strategy, which sometimes get stuck in local optima (Whitley, et al., 1994).

III. PROBLEM STATEMENT

4. Problem Statement

This chapter describes some limitations associated to the current GGGP methods and establishes the bases and theoretical framework to proceed to the solution proposals of the aforementioned issues. This chapter firstly describes the followed research methodology is described in subsection 4.1 together with the goals, hypothesis, assumptions and restrictions defined for this thesis. The subsection 4.2 defines the theoretical framework employed to describe and address the GGGP limitations. Then, in subsection 4.3 and the following subsections the limitations associated with GGGP initialization, variations operators and terminal optimization are presented and defined. Finally,

4.1. Research Methodology, Hypothesis and Goals

This research work has followed a hypothetico-deductive incremental model to define the main tasks of the research process. The hypothetico-deductive model is a scientific method that produces new knowledge based on the corroboration or falsification of previously defined hypothesis (Godfrey-Smith, 2009). Specifically, this model starts gathering information about a problem at hand. This data may be based on previous knowledge or it can be obtained by means of exploration or empirical procedures. Then, from the obtained knowledge, a hypothesis stating an explanation or solution to the given problem is elaborated. From the assumption of the corroboration this hypothesis, a set of predictions or consequences are deduced. Finally, a set of theoretical or empirical tests are performed to look for evidences that conflict with the above-mentioned predictions. If evidences are found, the hypothesis is rejected. Otherwise, the hypothesis is corroborated according to the predictions and the test performed until any evidence that falsify it is found.

An incremental approach has been applied to the hypothetico-deductive model. This way, the formulation of new hypothesis and the incorporation of new knowledge is based on previous knowledge and hypothesis corroboration or

falsification. Therefore, the followed hypothetico-deductive incremental model is defined as a set of iterations that comprises:

1. A process of theoretical and empirical research and data compilation of a problem.
2. The definition of facts, restrictions, hypothesis and goals for that problem.
3. The deduction or prediction of consequences given the formulated hypothesis.
4. The falsification or corroboration of the hypothesis by means of theoretical or empirical tests.

As result of each iteration, a set of validated hypotheses and solutions to the problem are obtained. These results incrementally form the knowledge base that enables the formulation of new hypothesis and that defines the contributions of this research work. The rest of this subsection describes the goals, hypothesis, assumptions and restrictions defined in the hypothetico-deductive research process.

4.1.1. Goals

G1. Analyze the state of the art of GP initialization methods and variation operators, in particular those related with GGGP and large search spaces involving large derivation trees.

G2. Analyze the state of the art of GP optimization methods when dealing with terminal symbols sets of high cardinalities.

G3. Analyze GGGP optimization performance when dealing with large derivation trees, and how the initial population distribution and the variations operators affect the performance.

G4. Analyze GGGP optimization performance when dealing with terminal symbols sets of high cardinalities.

G5. Determine the causes that produce the limitations of GGGP when dealing with large search spaces involving large derivation trees and terminal symbols sets of high cardinality values.

G6. Design a new population initialization method that generate genotypically uniform population samples.

G7. Design a new variation operator that enhances the exploitative behavior to actively guide the evolutionary optimization process.

G8. Define optimization guidelines to optimize terminal symbols according to the cardinality of the CFG terminal symbols set.

4.1.2. Hypotheses

H1. The GGGP performance sharply decreases when evolving large derivation trees or the cardinality of the set of terminal symbols increases.

H2. The GGGP performance reduction is caused by the population initialization and the variation operators.

H3. GGGP initialization methods do not generate representative samples of the search space.

H4. GGGP crossover operators are highly explorative and barely exploitative and they are not suitable for problems involving large search spaces with large derivation trees or set of terminals symbols with a high cardinality.

H5. GGGP variation operators based on population estimation of distributions (EDA) are too destructive and they likely eliminate promising solutions from the population.

H6. An initial population that firstly seeks uniformity in terms of numbers of recursions and then in terms of the distribution in the search improves the GGGP performance.

H7. A crossover based on population estimation of distribution with an enhanced exploitative behavior that avoids the EDA's generational replacement improves the performance of GGGP.

H8. Using local search for optimizing terminal symbols improve GGGP performance when dealing with CFGs with a large cardinality of the terminal symbol set.

H9. Partial optimization of terminal symbols is sufficient to determine individual goodness and guide the evolutionary process.

H10. Penalization and elimination of infeasible individuals reduce the performance of the optimization process.

4.1.3. Restrictions

R1. GGGP computational cost and speed convergence must be kept low.

R2. Metaheuristics optimization techniques do not usually obtain optimal solutions.

R3. The elaborated solutions must be simple and replicable.

4.1.4. Assumptions

A1. Recursive production rules are very important to generate main solution characteristics.

A2. The provided CFG are already are already proper grammars and there is no need to modify them to obtain an improved version. Otherwise, modifications on the CFG would imply modifications in the search and solutions space that may lead to unexpected optimization issues.

4.2. The Proposed Theoretical Framework

A theoretical framework is presented to provide new concepts and notation that help determining GGGP issues and to define the proposed solutions in the next section. Firstly, cardinality of CFG production rules and symbols, and derivation probabilities novel concepts are presented. Then, a new notation is presented to define sets and multisets of nonterminal and terminal symbols of derivation trees.

4.2.1. Cardinality

Given a CFG $G = (V, \sum, R, S)$ where $\sum$ is terminal symbols set, V is nonterminal symbols set, S is the axiom and R is the set of production rules of the form $A ::= \alpha$ such that $A \in V$ and $\alpha \in (V \cup \sum)^*$, the cardinality determines the number of different terminal string derivations that can be produced starting from a production rule or symbol. The cardinality of a production rule or symbol x is denoted as $|x|$.

Let $V_0, V_1, \cdots, V_n \subset V$ be defined as:

1. $V_0 = \{A \in V \mid \forall r \in R \; with \; r: A ::= s \; necessarely \; s \in \Sigma^*\}$, (2)

2. For $i = 0, 1, \dots, n$

$$V_{i+1} = \{A \in V \mid \forall r \in R \; with \; r: A ::= \gamma \; necessarely \; \gamma \in (\Sigma \cup V_i)^*\}, \quad (3)$$

according to the notation above,

1. Let $a \in \Sigma$, then the cardinality of the terminal symbol a is:

$$|a| = 1 \quad (4)$$

2. For $i = 0, 1, \dots, n$, let $r \in \mathrm{R}, \mathrm{r}: \mathrm{A} ::= \gamma, \mathrm{A} \in V_i$ and $\gamma = B_0, B_1, \dots, B_m$, then the cardinality of the production rule r, $|r|$, is defined as:

$$|r| = \prod_{k=0}^{m} |B_k| \quad (5)$$

3. For $i = 0, 1, \dots, n$, let $A \in V_i$ then the cardinality of a nonterminal symbol A, $|A|$, is defined as

$$|A| = \sum_{r \in R_A} |r|, \tag{6}$$

If we assume the CFG is unambiguous, the cardinality also equals the number of different terminal strings yielded from the production rule or symbol. The cardinality of non-recursive production rules is finite, while the cardinality of recursive production rules is infinite. In approaches with code bloat control where the derivation steps are limited, production rules cardinality is always finite. The cardinality of the axiom S of a CFG, $|S|$, is the cardinality of the universe of derivations $\mathbb{U}$ of that CFG. If the CFG is unambiguous, then $|S|$ also equals the cardinality of the CFG language.

4.2.2. Derivation Probabilities

Given a CFG, the probability of a derivation $A \overset{*}{\Rightarrow} \alpha, \alpha \in (V \cup \sum)^*$ which applies a sequence of production rules $\{r^i\}_{i=1}^n$, denoted as $P\left(A \overset{*}{\Rightarrow} \alpha, \{r^i\}_{i=1}^n\right)$, is defined as the probability of iteratively applying each of the n production rules in $A \overset{*}{\Rightarrow} \alpha$, P($\{r^i\}_{i=1}^n$). $P(\{r^i\}_{i=1}^n)$ is obtained by multiplying the probability of choosing each production rule r, in the i^{th} derivation step, $P(r^i)$, (see (7)). In the case of nonstochastic CFG, the probability of choosing each production rule r, $P(r)$, equals 1 divided by the number of production rules with the same left-hand side nonterminal symbol. For stochastic CFG, $P(r)$ is the custom probability assigned to the rule r.

$$P\left(A \overset{*}{\Rightarrow} \alpha\right) \equiv P(\{r^i\}_{i=1}^n) = \prod_{i=1}^{n} P(r^i) \tag{7}$$

If the CFG is unambiguous, then $P\left(A \overset{*}{\Rightarrow} \alpha, \{r^i\}_{i=1}^n\right)$ equals the probability of generating the string α from A independently of $\{r^i\}_{i=1}^n$ since there is only one possible derivation. Thus, the probability notation can be simplified to $P\left(A \overset{*}{\Rightarrow} \alpha\right)$. In particular, the probability of producing a sentence $s \in \sum^*$ from the axiom S using an unambiguous CFG, $P\left(S \overset{*}{\Rightarrow} s\right)$, equals the probability of generating s using that CFG, and it is denoted as $P(s)$. The average probability of any terminal string

derivation $\mathrm{A} \overset{*}{\Rightarrow} \mathrm{s}$ applying first the production rule r: $A::=\gamma, \gamma \in (V \cup \sum)^*$, , equals the probability of choosing r first divided by the number of possible terminal string derivations produced by r:

$$\langle P\left(\mathrm{A} \overset{*}{\Rightarrow} \mathrm{s}, r\right)\rangle = \frac{P(r)}{|r|} \tag{8}$$

If r: $A::=\gamma$ is recursive, then $|r| = \infty$ and, therefore, $\langle P\left(\mathrm{A} \overset{*}{\Rightarrow} \mathrm{s}, r\right)\rangle = 0$. Once more, if the CFG is unambiguous, then $\langle P\left(\mathrm{A} \overset{*}{\Rightarrow} \mathrm{s}, r\right)\rangle$ equals the average probability of generating the string s from r, $\langle P\,(\mathrm{s}, r)\rangle$.

4.2.3. Sets and Multisets of Nonterminal and Terminal Symbols of Derivation

Given a derivation $S \overset{*}{\Rightarrow} s$, $V_{S \overset{*}{\Rightarrow} s}$ represent the set of different nonterminal symbols in $S \overset{*}{\Rightarrow} s$ and $(V_{S \overset{*}{\Rightarrow} s}, m)$ is the multiset containing every nonterminal symbol in $S \overset{*}{\Rightarrow} s$ where m is a function $m: V_{S \overset{*}{\Rightarrow} s} \to \mathbb{N}_{\geq 1}$ with $\mathbb{N}_{\geq 1} = \{1,2,3 \ldots\}$ that represents the number of occurrences of each nonterminal symbol $A \in V_{S \overset{*}{\Rightarrow} s}$.

4.3. Population Initialization Bias

The GGGP initial population distribution is difficult to control since it strongly depends on the CFG definition. The initial distribution is mainly shaped by the production rule probabilities and cardinalities. There are two main causes of the population distribution bias: the cardinality inequality within a nonterminal production rule set and the low probability of generating deep derivation trees.

On one hand, different cardinalities values within the same nonterminal symbol production rules set shape population initialization in different ways. According to (8), the average probability of generating any terminal string s from A depends on the cardinalities of each production rule included in $A \overset{*}{\Rightarrow} s$. Let $A::=\gamma_i$ and $A::=\gamma_j$ be two production rules of R_A with $i \neq j$. According to (8), if $P(A::=\gamma_i)$ and $P\left(A::=\gamma_j\right)$ are similar, and $|A::=\gamma_i|$ is higher than $\left|A::=\gamma_j\right|$, then it is less likely to obtain, on average, a terminal string s starting from $A::=\gamma_i$, than starting

from $A::=\gamma_j$. Therefore, the population initialization is biased to those production rules with lower cardinalities. On the other hand, according to the (7), the probability of a derivation decreases as the number of derivation steps required increases. Since this value is obtained from a product of probabilities, it tends to 0 as the number of production rules in $\{r^i\}_{i=1}^n$ increases. This fact favors the production of smaller derivation trees. Therefore, most of the search space generated by recursive CFGs is unreachable during the initialization since long derivations involving the same recursive production rule are unprovable.

Most of the population initialization methods are devoted to avoid code bloat and seek uniformity. The probability of selecting any production rule given a nonterminal symbol is the same on initialization methods that employ nonstochastic CFGs. However, this setup does not ensure any population uniformity. Stochastic CFGs are used to shape the production rules selection and seek custom distributions. A proper tune of the production rules probabilities may help to grant uniformity in non-recursive CFGs. However, manually assigning and tuning these probabilities can be tedious and inaccurate. In the case of recursive CFGs, tuning the production rule probabilities is insufficient since these probabilities are static during each individual generation. An example with a right-recursive CFG ($G_{Recursive}$) is presented. The language generated by $G_{Recursive}$ comprises any n-length string with the terminal symbol 1; in regular form: $L(\mathrm{G_{Recursive}}) = 1^n$.

$$
\begin{aligned}
&G_{Recursive} = (V, \Sigma, R, S)\\
&V = \{Recursive\}\\
&\Sigma = \{1\}\\
&R = \{\\
&\qquad r_1 \quad S ::= Recursive\\
&\qquad r_2 \quad Recursive ::= 1\ Recursive\\
&\qquad r_3 \quad Recursive ::= 1\\
&\}
\end{aligned}
\tag{9}
$$

According to (7), and considering that $G_{Recursive}$ is unambiguous,

$$P\left(S \overset{*}{\Rightarrow} 1^n, \{r^i\}_{i=1}^n\right) = P\left(S \overset{*}{\Rightarrow} 1^n\right) = \prod_{i=1}^{n} P(r^i)$$

In this case, $r^1 = r_1$, $r^2 = r^3 = \cdots = r^n = r_2$ and $r^{n+1} = r_3$. Then,

$$P\left(S \overset{*}{\Rightarrow} 1^n\right) = P(r_1)P(r_2)^{n-1}P(r_3),$$

where $\mathrm{P}(r_1)$ is 1, $\mathrm{P}(r_2) = \mathrm{P}(r_3) = 0.5$. Therefore, $P(1^n, S) = 0.5^n$. Consequently, the probability of getting long words tends to 0. An experiment using a standard population initialization method on $G_{Recursive}$ is performed to empirically show the population distribution. A study on sentence length n is performed to analyze the population initialization capability to iteratively select the recursive production rules. Every time r_2 is applied, the resulting sentence length is increased by one. Figure 9 shows the sentence length probability distribution of 1000 generated individuals by the regular GGGP population initialization (Table 8). The abscissa axis represents sentence length, and the ordinate axis the probability of a sentence with length X to be generated. According to theoretical results, the chart shows a strong bias in the population initialization that tends to generate short sentences that rarely exceed length 8. Likewise, almost 50% of the generated derivation trees do not even include r_2, so the length of their resulting sentences is 1.

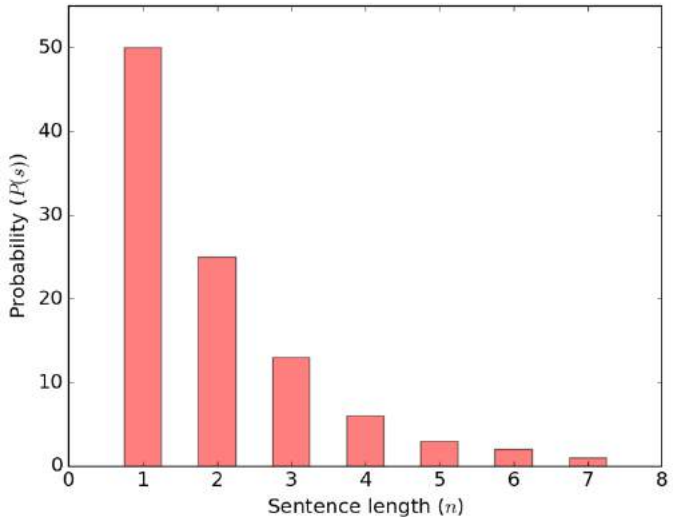

Figure 9: Probability distribution using regular population initialization for $G_{Recursive}$. Maximum sentence length: 50.

A stochastic CFG can be applied to bias population initialization with the goal of increasing the average sentence length. However, even assigning high probabilities to the recursive production rules, short sentences are more likely to be generated. Figure 10, obtained from (7) and $G_{Recursive}$, shows the probability of obtaining sentences with a random length n, $P(s = 1^n)$, as the probability assigned to r_2, $P(r_2)$, varies. High $P(r_2)$ values increase the probability of generating long sentences. However, the probability of generating short sentences is still higher since $\prod_{i=2}^{n} P(r_2^i)$ tends to 0 as n increases, $\lim_{n\to\infty} \prod_{i=2}^{n} P(r_2^i) = 0$. As Figure 10 shows, sentence probability distribution is strongly biased towards short sentences for $P(r_2) < 0.8$. The average sentence length is below 4 ($n < 4$), and the probability of generating sentences of length $n > 6$ is almost zero. As $P(r_2)$ approaches 1, probability distributions would be more similar to a uniform probability distribution, but at the expense of generating very long sentences.

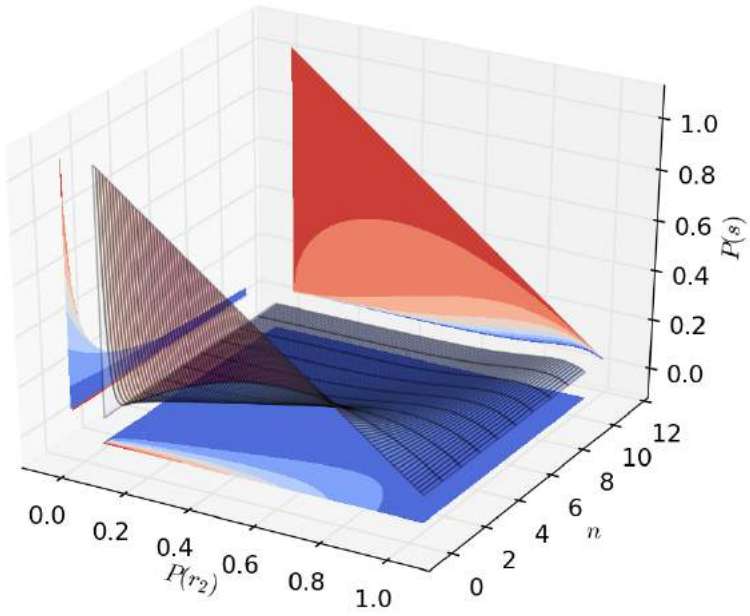

Figure 10: Probability $P(s)$ of obtaining a sentence s of length n with $P(r_2)$ probability of choosing the recursive rule r_2 from $G_{Recursive}$

Very long sentences are not usually suitable for GGGP since they produce code bloat. Code bloat is usually solved by setting a maximum derivation tree depth bound. Such solution with $P(r_2)$ close to 1 would produce strongly biased initial populations where most of the individuals reach maximum depth bound. That is,

most of the sentences reach the maximum length n. Figure 11 shows an example of a population initialization with $P(r_2) = 0.99$ and a sentence length bound of 10. The probability of generating a sentence of length 10 is 0.91 while the rest of the lengths sum a probability of 0.09. Consequently, current approaches that combine GGGP and EDA are not suitable for these kinds of problems since optimization process is usually driven by a population initialization method with a stochastic CFG.

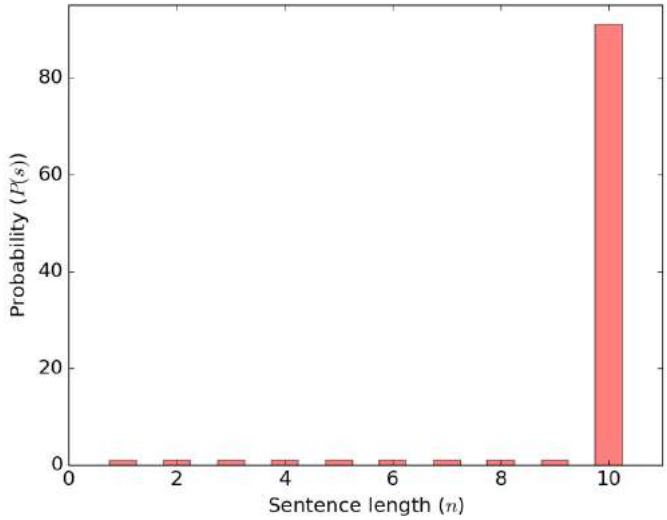

Figure 11: Probability $P(s)$ of obtaining a sentence s of length n with $P(r_2) = 0.99$

The cardinality of production rules is another cause for the initialization bias. Population initialization can be strongly biased when the cardinalities of the productions rules with the same left-hand side nonterminal symbol are highly different. This happens in both: recursive and non-recursive CFGs. For the example given $G_{Recursive}$ and according to (8):

$$\langle P\left(Recursive \overset{*}{\Rightarrow} s, r_2\right)\rangle = \langle P\,(s, r_2)\rangle = \frac{P(r_2)}{|r_2|} = 0;$$

$$\langle P\left(Recursive \overset{*}{\Rightarrow} s', r_3\right)\rangle = \langle P\,(s', r_3)\rangle = \frac{P(r_3)}{|r_3|} = 0.5,$$

as $|r_2| = \infty$, $|r_3| = 1$ and $P(r_2) = P(r_3) = 0.5$. Consequently, a terminal string s' generated from production rule r_3 is more likely to be generated, on average, than another terminal string s from r_2. Setting a depth threshold to control code bloat

in $G_{Recursive}$ will make $\langle P\,(s, r_2\,)\rangle > 0$. However, $\langle P\,(s, r_2\,)\rangle \leq \langle P\,(s', r_3\,)\rangle$ is still satisfied and, therefore, the initialization is also still biased.

4.3.1. Resolution Guidelines

In order to reduce these limitations, a novel population initialization method that generate representative samples of the CFG is presented: the grammatically uniform population initialization (GUPI). This population initialization method is designed to generate new individuals that are first uniformly distributed in terms of number of recursive derivations and then uniformly distributed in the search space generated by the CFG. An important performance improvement is expected since initial populations would me more representative of the search space. Thus, the exploratory effort would be considerably reduced.

4.4. Limitations of Variation Operators

Variation operators produce new individuals based on the characteristics of a set of individuals obtained by the selection operator. Selection operator chooses individuals encoding promising solutions to help the variation operators to seek new better adapted individuals. Variation operators are intended to lead the evolutionary process towards new promising individuals so that the obtained offspring improves or maintains its parents' fitness. However, this task is not easily complied and in fact many variations operators do not achieve it. Two different popular variation operator approaches are analyzed in this section to show how they perform in the task of leading the evolutionary process: a genetic crossover and an estimation of distribution variation approach for GGGP.

4.4.1. The Crossover Operator Exploration Problem

The GGGP crossover is a genetic based variation operator that swap subtrees of parents' derivation trees to produce a new offspring. The most popular crossover operator for GGGP is Whigham crossover (WX). Although, this operator produces an offspring based on parents' genotype, depending on the selected crossover nodes, the resulting offspring might not be similar to parents' individuals. According to

this fact, the optimization may not focus on exploiting promising individuals, but instead it focuses on exploring new areas of the search space. Thus, the evolutionary process may show an erratic behavior and some difficulties to progress towards promising solutions. These issues are related to the genotypic (syntactic) locality of crossover operators, that is, the ability of the crossover operators to perform small changes to the genotype.

WX pursues the genotypic locality crossover nodes must contain the same nonterminal symbol. Thus, the swapped subtrees are rooted with the same nonterminal symbol. The genotypic locality is then strictly dependent on the CFG definition. If the cardinality of a nonterminal symbol is small, then genotypic locality is likely kept for that symbol since small changes are usually expected from nonterminals with a low cardinality. Otherwise, locality is less probable to be kept. According to this, nonterminal symbols that lead to recursive derivations barely keep genotypic locality since their cardinality is infinite. Although the size bound is not a feature of WX crossover, it can be part of the crossover operation affecting the resulting offspring. In addition to code bloat prevention, size bounds also help keeping locality by delimiting the search space. However, when size bounds are high, the probability of keeping locality is still low for nonterminal symbols that lead to recursive derivations.

Given two derivations $S \overset{*}{\Rightarrow} s$ and $S \overset{*}{\Rightarrow} s'$ as parents, and their corresponding multisets $\left(V_{S\overset{*}{\Rightarrow}s}, m\right)$ and $\left(V_{S\overset{*}{\Rightarrow}s'}, m'\right)$, the cardinality of the offspring derivations set that can be produced by by WX, $X_{\left(S\overset{*}{\Rightarrow}s, S\overset{*}{\Rightarrow}s'\right)}$, is determined by:

$$\left|X_{\left(S\overset{*}{\Rightarrow}s, S\overset{*}{\Rightarrow}s'\right)}\right| = \sum_{A\in\left(V_{S\overset{*}{\Rightarrow}s}\cap V_{S\overset{*}{\Rightarrow}s'}\right)} m(A)m'(A) \tag{10}$$

where $\left(V_{S\overset{*}{\Rightarrow}s} \cap V_{S\overset{*}{\Rightarrow}s'}\right)$ represent the set of nonterminal symbols that can be selected as crossover nodes, and $m(A)$ and $m'(A)$ the number of occurrences a nonterminal symbol A in the derivations $S \overset{*}{\Rightarrow} s$ and $S \overset{*}{\Rightarrow} s'$ respectively. The number of the different offspring derivations increases according to the number of different feasible crossover points of parents' derivations. The probability of generating new

derivation trees that do not preserve the genotypic locality increases when the number of crossover nodes of nonterminal symbols with large cardinality increases. Figure 12 represents the number of different derivations that can be produced by WX according to the number of crossover points of the two parents generated with $\mathsf{G}_{\mathsf{Recursive}}$ (9). Since only one nonterminal symbol ($Recursive$) can be selected as crossover point in $\mathsf{G}_{\mathsf{Recursive}}$ derivations then, from (10) it is obtained that $\left|X_{\left(S\overset{*}{\Rightarrow}1^n, S\overset{*}{\Rightarrow}1^{n'}\right)}\right| = m(Recursive)m'(Recursive)$. Note that $\mathsf{G}_{\mathsf{Recursive}}$ search space is infinite since there is a recursive production in its production rule set, r_2. Then, the histogram only represents a small range of the possible crossover points number, from 1 to 10.

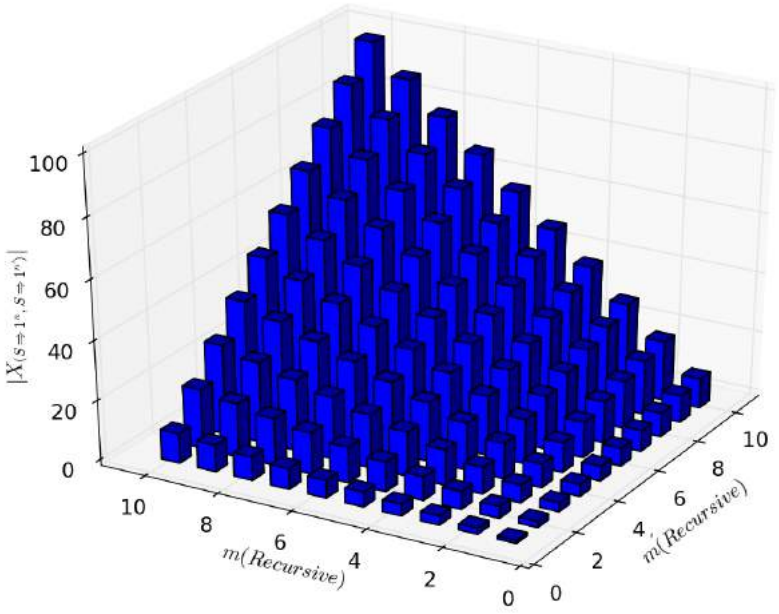

Figure 12: Number of derivation trees that can be produced by WX according to the number of the occurrences of nonterminal $\mathsf{Recursive}$ *in each parent derivation,* $m(Recursive)$ *and* $m'(Recursive)$.

The number of different derivations increases as the number of possible parents' crossover points increases. The growth is linear when one of the parents' possible crossover points remains constant and quadratic when both parents' crossover points increases at the same pace. Note that the number of different possible derivations is already 100 when parents possess 10 possible crossover points each.

To determine whether WX keeps genotypic locality for $G_{Recursive}$, the difference of the sentences length between the offspring's sentences and the parents' sentences is calculated. If the difference of sentences length is not higher that a bound l, then WX keeps the genotypic locality of grade l. Figure 13 to Figure 16 show the genotypic locality preservation rates of the offspring according to the locality grades $0,1,2$ and 3, and the sentence length of parents' derivation. In general terms, it is shown that the locality is not likely preserved by WX. When locality grade is set to low values, the locality preservation rate is very low and almost 80% of the offspring derivations do not keep it. In the case of $l = 0$, Figure 13 shows that independently of the parents' sentence length, the genotypic locality is only preserved by less than 20% of the offspring. For higher values of l, the locality preservation rates rise. However, high rates appear when the range of possible lengths of the offspring sentences is comparable to the value l. In the case of $l = 3$, Figure 16 shows high rates of locality preservation when at least one of the parents' sentence length equals 5 since the sentence length is close to l. However, when the sentence length increases, the locality preservation decreases and only 50% of the offspring keeps it on average. Note that genotypic locality preservation rate provides an approximated measure of WX exploration-exploitation when using $G_{Recursive}$ where l defines the bounds between the exploration and the exploitation. An exploitation behavior appears when locality is kept, and an exploration behavior, when it is not.

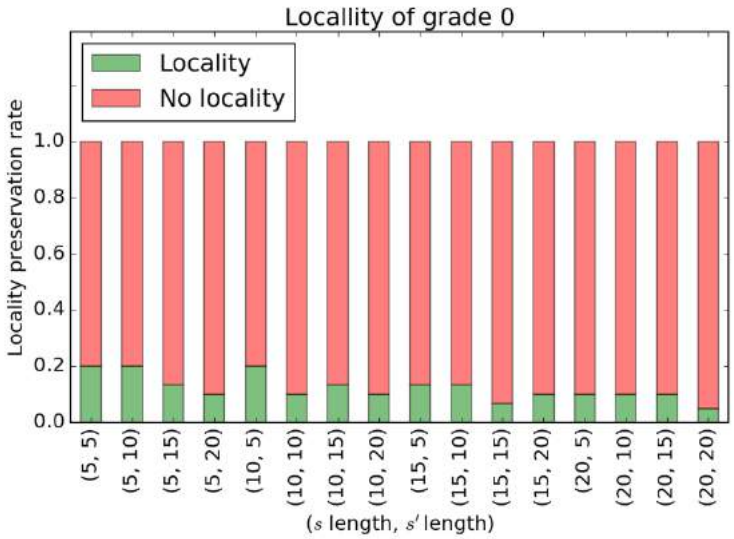

Figure 13: Genotypic locality preservation rate for locality grade $l = 0$ according to the parents' sentences length.

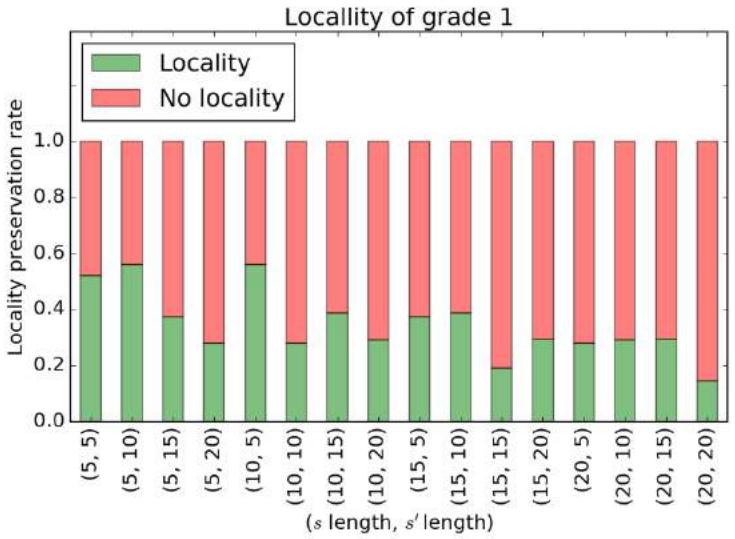

Figure 14: Genotypic locality preservation rate for locality grade $l = 1$ according to the parents' sentences length.

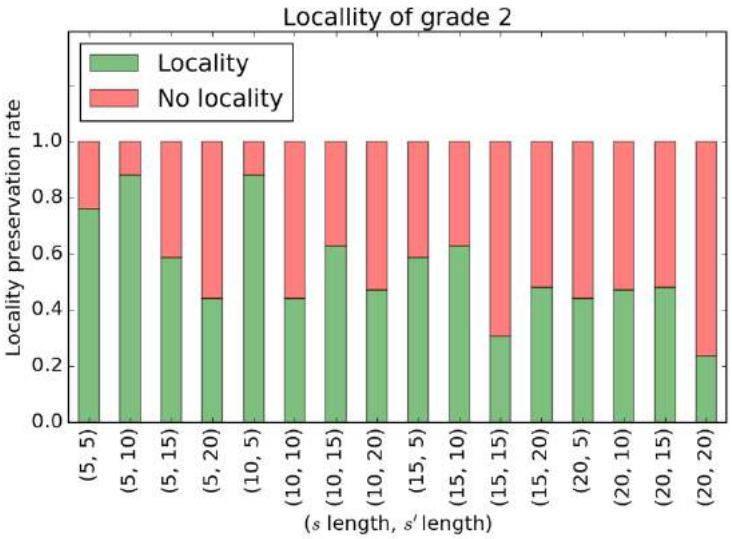

Figure 15: Genotypic locality preservation rate for locality grade $l = 2$ according to the parents' sentences length.

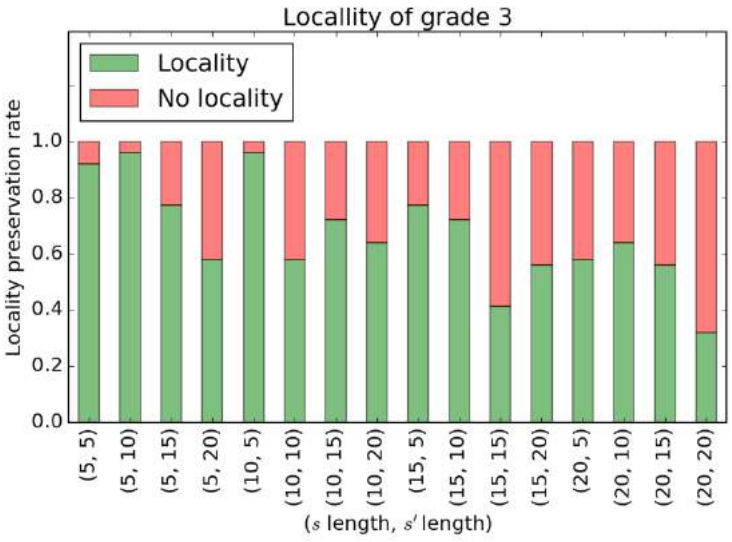

Figure 16: Genotypic locality preservation rate for locality grade $l = 3$ according to the parents' sentences length.

4.4.2. The Diversity Loss of Estimation of Distribution Algorithms

EDAs have been proposed as a promising alternative to genetic evolutionary algorithms such as GA, GP and GGGP. In general terms, EDAs outperforms most of the genetic approaches. Compared to them, the EDA optimization process shows a fast convergence to optimal solutions. However, the EDA approaches to GGGP may show a performance drawback caused by the probabilistic model and the

generational replacement. In particular, EDAs shows a reduction of population diversity and excessively exploitative behaviour.

Conversely to genetic evolutionary approaches, EDAs do not exchange genotype segments to produce new generations of individuals. EDAs estimate a probability distribution of a subset of promising individuals of the population to later produce a new population according to this estimated distribution where more extended characteristics are more likely reproduced. In general terms, genetic evolutionary optimization processes present a more exploratory behaviour than EDAs, which show a more reliable reproduction of parents' genotype. The genotypic locality of the variation operators determines the difference of the exploratory-exploitative trade-off in GGGP approaches. Approaches with an enhanced locality preservation exposes a more exploitative behaviour. For regular GGGP, the locality is enforced by ensuring the crossover operator only swaps subtrees with the same nonterminal in the root node. In EDAs approaches, there is no subtree recombination. New individuals are generated from the root to the leaves following the estimation distribution of parents' derivation trees. Therefore, the genotypic locality is stricter for EDAs approaches and consequently, EDAs shows a more exploitative behaviour that would reduce the convergence time, but also increase the probability to converge to local optima.

The performance of the EDAs' optimization process mainly relies on the probabilistic model abstraction. A more explicit representation of the population distribution tends to produce individuals that are likely similar or equal to previous promising individuals, so that, the exploitative behaviour is boosted. However, it is more probable to converge to local optima since spatially close promising areas of the search space are unlikely explored. On the contrary, a more abstract representation of the population distribution facilitates the production of new promising individuals that are spatially close in the search space. Then, the optimization process will show a more exploratory behaviour that will reduce the probability to converge to local optima to the detriment of the convergence speed. The EDAs approaches for GGGP inherently provide an explicit representation of

the population distribution that reduce the exploratory behaviour of the optimization process.

Figure 17 to Figure 20 show a comparison of the fitness evolution of the best-fit population individual in a GGGP minimization optimization problem using WX, and an EDA approach to GGGP when looking for derivation trees of $G_{Recursive}$ (9) with 5, 10, 20 and 50 recursive derivations. WX chooses 2 individuals to produce 2 new individuals. EDA chooses 50% of the population to produce a new whole population. Both approaches employ the tournament selection to improve the population diversity. No mutation or immigration are applied to reduce the random component of the evolutionary process and facilitate the comparison. Population size is fixed to 100. The GGGP fitness function is the hamming distance between the word encoded in the target derivation tree and the word of the evaluated derivation tree. The derivations words are compared, so that, for each position of the words, the fitness value is increased by one every time the elements in that position are different. If the lengths of the words are not the same, the difference of the lengths is summed to the fitness value since the last elements of the longest word cannot be compared. The evolutionary process seeks to minimize the fitness value. The evolutionary process converges if it finds the target derivation tree or it accomplishes 50 generations without improving the average population fitness. The abscissa axis shows the best-fit population individual fitness while the ordinate axis the generation/iteration number of the evolutionary process. While WX is able to recombine parents' derivation trees to produce new derivation trees with different number of recursive derivations that improve the fitness of the best population individual, EDA cannot obtain derivation trees with different number of recursive derivations, and thus, the fitness of the best population individual remains constant. According to this, the application of additional methods, such as mutation, that increase the exploratory behaviour to search for new individuals may be even more important in EDA than in regular GP and GGGP approaches.

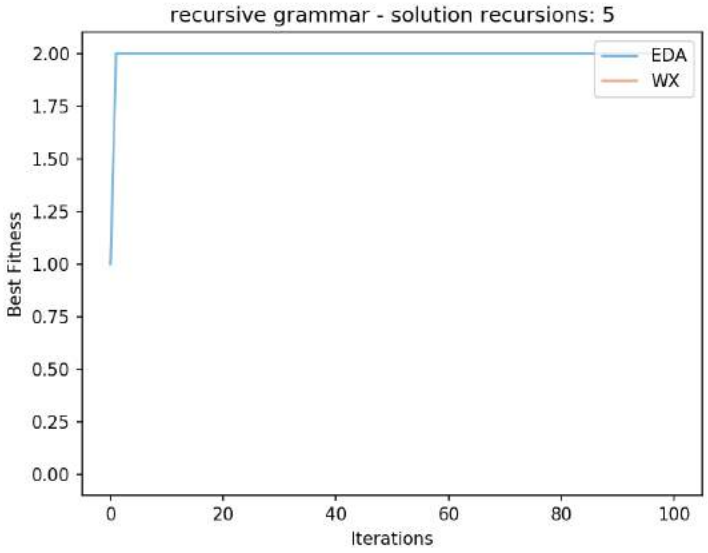

Figure 17: Comparison of fitness evolution of the best-fit population individual when looking for a derivation tree of $G_{Recursive}$ with 5 recursive derivations according to the optimization approach: GGGP with WX, and a EDA approach to GGGP

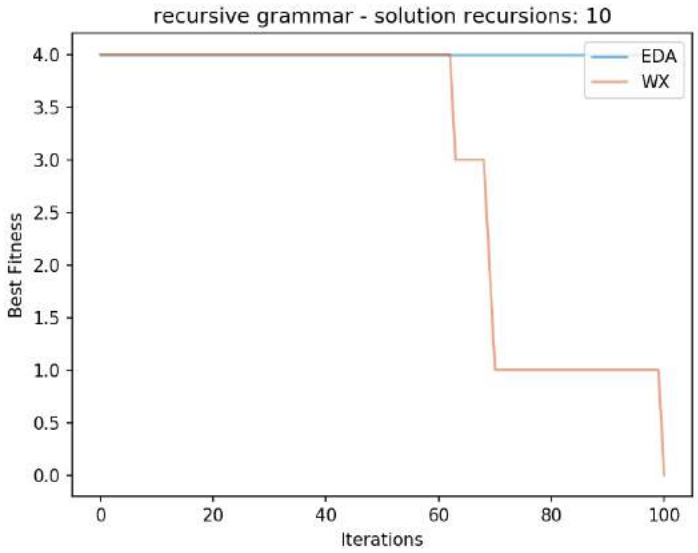

Figure 18: Comparison of fitness evolution of the best-fit population individual when looking for a derivation tree of $G_{Recursive}$ with 10 recursive derivations according to the optimization approach: GGGP with WX, and a EDA approach to GGGP

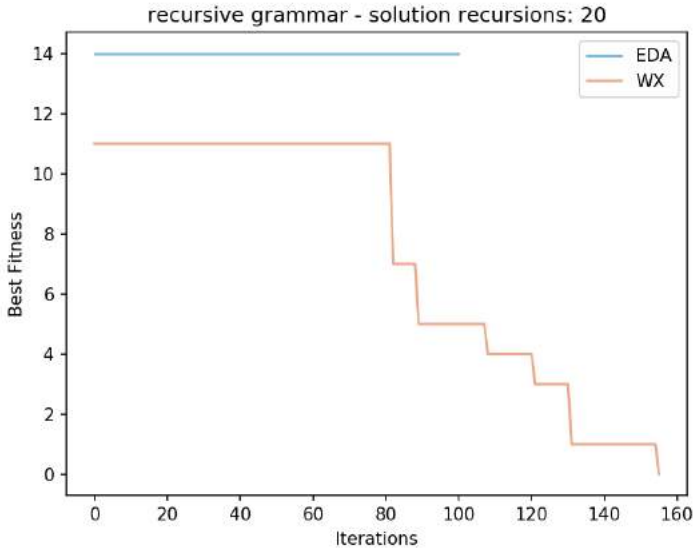

Figure 19: Comparison of fitness evolution of the best-fit population individual when looking for a derivation tree of $G_{Recursive}$ with 20 recursive derivations according to the optimization approach: GGGP with WX, and a EDA approach to GGGP

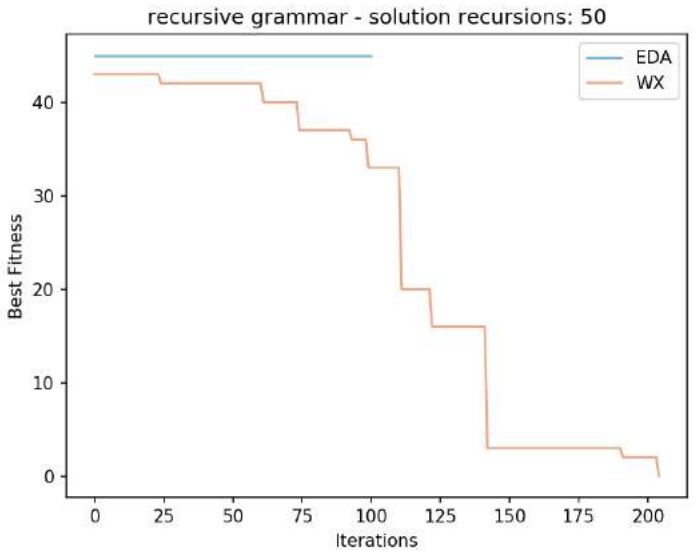

Figure 20: Comparison of fitness evolution of the best-fit population individual when looking for a derivation tree of $G_{Recursive}$ with 50 recursive derivations according to the optimization approach: GGGP with WX, and a EDA approach to GGGP

Some EDA approaches have implemented additional tools to provide an evolutionary environment that reproduce the building block theory of genetic algorithms. In general terms, this technique has improved EDAs performance. However, this is mainly due to the fact that the exploratory behaviour of EDA's

evolutionary process is increased. Actually, the building block relocation is based on the assumption that building blocks provide an improvement of fitness independently of where they are located in the genome, but this is problem dependent, so it is not always complied. Subsequently, better exploratory techniques must be provided to improve the evolutionary process and avoid basing the exploratory behaviour on relocating already explored building blocks.

Both, genetic evolutionary algorithms and EDAs are designed to maintain the characteristics of most promising individuals from generation to generation. This is achieved in the case of GAs since replacement operator rarely choose promising individuals to be replaced by the new offspring. This way, they will be likely reutilised in following generations. On the other hand, EDAs algorithms are designed to replace the whole population by a new generation following a probabilistic model. Then, only those characteristics that are widely spread in the population are more likely transferred to the new population. However, the characteristics of the most promising individuals that are not so widely spread in the population are transferred with a lower probability. Consequently, the probabilistic model iteratively focuses on more extended characteristics within the selected population individuals. When using explicit representation of the population distribution, this feature can produce diversity loss since those promising individuals that are not produced in the next generation, cannot be reproduced in following generations. Figure 17 shows an example of diversity loss when looking for a derivation tree with 5 recursive derivations. In the first iteration, the best-fit population individual, with a fitness value of 1, is not reproduced, and consequently, it is never produced again. Since population initialization is biased and it likely produces derivation trees with few recursive derivations (close to 0), more derivation trees with fewer recursive derivations will be produced because the probabilistic model increases the probability of small trees. To avoid the elimination of promising characteristics, incremental EDAs, which probabilistic model is incrementally updated, has been proposed. However, this feature is not enough to override the pressure of widely spread characteristics, and those

promising characteristics will finally vanish if they are not replicated in following generations.

4.4.3. Resolution Guidelines

To reduce some of the presented limitations, a hybrid crossover operator based on the estimation of distribution is presented: the estimation of distribution crossover (EDX). To such aim, the CFG must be extended to represent the search space in a graph-like structure that stores the probabilities of the estimated distribution, called context free grammar expansion (CFGE). The EDX is designed to generate new individuals containing the most representative characteristics of a subset of previously selected promising individuals of the population, and thus, adequately guide the evolutionary process. Contrary to the EDAs generational replacement, in the EDX only a small subset of individuals is produced and replaced each evolutionary iteration. This setup is intended to reduce the diversity loss and consequently avoid the premature convergence to suboptimal solutions.

4.5. Terminals Optimization Limitations

GGGP is an evolutionary algorithm that generates variable size programs (derivation trees). The elements of these programs, terminals and nonterminals symbols, are chosen from a set of possible elements defined in the CFG. When a nonterminal symbol $A \in V_o$ produces large number of different terminal symbols, that is, when $|A|$ reach high values, GGGP algorithms may result inefficient to find the terminals' combination that better solves the problem. The set of production rules with $A \in V_o$ in the left-hand side of each production, where $|A|$ reaches high values, are called critical terminal production rules (CTR) and its terminal symbols, critical terminal symbols (CT). Let us suppose that a GGGP algorithm has already found a partial solution $S \overset{*}{\Rightarrow} s$ for a problem at hand where the structure of the program and all the terminals but one are already fully optimized. If this last terminal is produced by a critical terminal production, the resolution of the problem may slow down or even become computationally unreachable.

There are two common ways to obtain this last terminal, a, of the partial solution $S \overset{*}{\Rightarrow} s$: by mutation or crossover operations. Mutation implies a random change in a random element of the derivation tree. To simplify the problem, let's assume that the partial solution is selected with probability 1 (best possible case) since it has a very good fitness value. The probability that mutation operator choose the nonterminal symbol A that produces a is $\frac{1}{\left|\left(V_{S\overset{*}{\Rightarrow}s}, m\right)\right|}$, where $\left|\left(V_{S\overset{*}{\Rightarrow}s}, m\right)\right|$ is the number of different crossover points of the derivation tree $S \overset{*}{\Rightarrow} s$. The mutation operator chooses a random terminal from the set of possible terminals of the critical terminal productions. So that, from (7), the probability of choosing the right terminal a from the set of possible terminals is $P(A \Rightarrow a) = P(A ::= a) = \frac{1}{|A|}$ where $|A|$ is the cardinality of A. In this case, the number different terminals that can be produced from A since $A \in V_o$. To sum up, the probability to obtain a successful mutation that obtains the last terminal a is defined by $\frac{1}{\left|\left(V_{S\overset{*}{\Rightarrow}s}, m\right)\right|}\frac{1}{|A|}$.

The crossover operator performs the genome recombination of different individuals. The GGGP crossover chooses one random crossover point from each parent derivation tree and swaps their subtrees. To obtain a successful change that obtains a, the selection operator must choose the partial solution individual $S \overset{*}{\Rightarrow} s$ and another individual from the population that contains a. To simplify the problem, let us assume that the partial solution and the other individual containing a are selected with probability 1 (best possible case). Once the parents are selected, there are two possible variations: The crossover point is firstly selected in the partial solution or in the second parent derivation tree. When the crossover point is firstly selected in the partial solution derivation tree, the crossover operator must select the nonterminal symbol A that produces a. This probability is again $\frac{1}{\left|\left(V_{S\overset{*}{\Rightarrow}s}, m\right)\right|}$. The selection of other nonterminal symbols that may contain a subtree that solves the problem is obviated since the probability to find another subtree with the same structure, including a is even lower. All feasible crossover points of the second parent contain the nonterminal symbol, A, which might produce a. The probability

of producing a on any selected crossover point is $\frac{1}{|A|}$. Therefore, the probability to execute a crossover that solves the problem is less than $\frac{1}{\left|\left(V_{S\Rightarrow^{*}s},m\right)\right|}\frac{1}{|A|}$ when the crossover point is firstly selected in the partial solution derivation tree.

Comparably, if the crossover point is firstly selected in the second parent derivation tree, the probability of selecting a crossover point containing A nonterminal symbol equals $\frac{m'(A)}{\left|\left(V_{S\Rightarrow^{*}s'},m\right)\right|}$ since there is $m'(A)$ in $S\overset{*}{\Rightarrow}s'$. Once selected the nonterminal symbol A, the probability of that nonterminal symbol produces a is $\frac{1}{|A|}$. Then, the probability of selecting the nonterminal symbol A that produces a in the partial solution derivation tree is $\frac{1}{m(A)}$. Therefore, the probability of producing a successful change when firstly selecting the crossover point in the second parent derivation tree is less than $\frac{m'(A)}{\left|\left(V_{S\Rightarrow^{*}s},m\right)\right|}\frac{1}{|A|}\frac{1}{m(A)}$. Finally, the total probability of producing a successful change in a crossover operation is less than $\frac{1}{2}\frac{1}{\left|\left(V_{S\Rightarrow^{*}s},m\right)\right|}\frac{1}{|A|}+\frac{1}{2}\frac{m'(A)}{\left|\left(V_{S\Rightarrow^{*}s},m\right)\right|}\frac{1}{|A|}\frac{1}{m(A)}$.

Both mutation and crossover operations are applied according to a predefined probability. Consequently, the previous probabilities must be multiplied by the mutation and crossover probabilities reducing even more the probability of a successful change. Therefore, the probability of obtaining a specific critical terminal symbol a by applying either mutation or crossover, even when starting from a quite good solution near the optimum one, is very low.

In the case of EDAs' approaches to GGGP where the partial solution is already a widely-spread solution in the population, the probability to obtain a successful change is determined by the probability encoded in the probabilistic model to generate the right terminal symbol a from the nonterminal symbol A that produces it. There are two possible cases: the probability to generate a from A is either 0 or not. In the first case, a can be only produced by the mutation operator. Then the probability to obtain a successful change is again $\frac{1}{\left|\left(V_{S\Rightarrow^{*}s},m\right)\right|}\frac{1}{|A|}$. In the second case,

the EDA approach will shortly produce a derivation tree with that desired terminal symbol a.

A set of experiments have been performed to show the limitations of GGGP when trying to obtain sentences of a specific length containing nonterminal symbol of a finite terminal symbol set. To such aim, the CFG G_{CT} is designed to show the limitations when working with critical terminals. The production rule r_1 determine the length of the sentences while the rules r_3 and the following rules the CTs.

$$
\begin{aligned}
&G_{CT} = (V, \Sigma, R, E)\\
&V = \{CT\}\\
&\Sigma = \{a, b, c, d, e, \dots\}\\
&R = \{\\
&\qquad r_1 \quad E ::= E\ E\\
&\qquad r_2 \quad E ::= CT\\
&\qquad r_3 \quad CT ::= 1\\
&\qquad r_4 \quad CT ::= 2\\
&\qquad r_5 \quad CT ::= 3\\
&\qquad r_6 \quad CT ::= 4\\
&\qquad r_7 \quad CT ::= 5\\
&\qquad r_8 \quad CT ::= \dots\\
&\}
\end{aligned}
\tag{11}
$$

Sentences from length 2 to 19 are searched. The cardinality of the terminal symbol set equals the length of the sentence searched in the experiment, so that, when looking for a sentence of length 5, the terminal symbol set contains {1,2,3,4,5}. GGGP applies the regular population initialization (Table 8) and a WX crossover operator. 2 new individualst are generated every generation. Mutation is applied with probability of 0.02. Population size is fixed to 50. 50 executions are run for each sentence. The fitness function is the sum of the absolute value of the difference between each element of the evaluated sentence with the element in the same position of the target sentence. If the lengths of the sentences are different, the length difference multiplied by the cardinality of the terminal symbol set is added to the fitness value. The evolutionary process converges when the target sentence is obtained or when the population average fitness does not improve for 50 generations. Table 9 shows the statistical description of GGGP results. The value

n determines the sentence length and the cardinality of the terminal symbol set. GGGP does not always obtain the target derivation independently of the value of n. From $n = 0$ to $n = 5$, GGGP usually obtains the target derivation tree. However, for $n > 6$, it starts showing some difficulties to obtain it. Moreover, for the $n = 11$ and $n > 14$, GGGP is not able to find the target derivation tree in any of the 50 runs. Other optimization techniques that take advantage of the order relation of the terminal symbols have been applied as a local search technique to optimize the terminal symbols. For these cases, the target sentence is obtained for all sentences. However, this is achieved at the expense of increasing the computational cost.

Table 9: Statistical description of the results of GGGP optimization algorithm when looking for sentences of different length n containing terminals of a terminal symbol set with cardinality n

n	Avg.	SD.	Min	max
2	0.100000	0.303046	0	1
3	0.300000	0.505076	0	2
4	0.280000	0.496518	0	2
5	0.700000	0.735402	0	3
6	1.280000	1.246055	0	4
7	1.620000	1.496799	0	5
8	2.300000	1.693324	0	6
9	3.340000	1.709637	0	7
10	3.620000	2.13704	0	8
11	4.900000	2.557502	1	9
12	5.620000	2.202874	0	10
13	6.520000	2.409505	0	10
14	7.840000	2.510061	0	12
15	7.92000	2.79825	1	13
16	9.580000	2.295426	4	13
17	10.500000	2.858928	2	15
18	11.480000	2.604862	3	16
19	12.160000	2.613622	6	16

4.5.1. The Feasibility of the Encoding Scheme

The local optimization of terminal symbols may imply considering additional requirements related to the definition of the encoding scheme. General-purpose encoding schemes do not usually consider problem-dependent constraints and the generation of infeasible individuals is probable. There are three common strategies to deal with the generation of infeasible individuals: elimination of infeasible

individuals, penalty functions and repair operators. In general terms, the application of these strategies produce a performance decrease. The elimination of infeasible individuals increases the computational cost since more individuals must be generated. The penalty function implies the presence of infeasible individuals in the population that reduce the population diversity and the evolutionary process may lead to premature convergence. Finally, the repair operators perform an additional operation once individuals are generated to enforce the feasibility of the encoded solutions, increasing the computational cost. Moreover, all of them may bias or randomize the evolutionary process due to the unrelated changes performed to the population or the individuals genotype.

In particular, there is a common constrain in optimization problems that require that the sum of the solution components must equals a predefined constant. The penalty functions or the elimination of infeasible individuals are not suitable to solve this constrain problem since almost every individual generated is infeasible, and consequently, all of them are penalized or eliminated, halting the evolutionary process. Repair operators can be applied to obtain feasible solutions and avoid the presence of infeasible ones in the population. However, the application of these operators has a high computational cost since every element of the solution must be modified. Moreover, the repair operator may produce unexpected changes to the offspring's genotypes so that they are no longer related to their parents. Thus, the optimization process may show an erratic evolution.

4.5.2. Resolution Guidelines

A novel asynchronous endosymbiotic co-optimization technique and a new encoding scheme are presented to overcome the presented limitations: the endosymbiotic co-optimization (ECO) method and the partition based encoding scheme (PBES). The ECO method is designed to provide an asynchronous mechanism for parallel optimization of the derivation tree structures and their terminal symbols. The terminal optimization method is designed to provide estimates of the individuals' fitness that guide the evolutionary process, so that, the evolutionary process is not

halted. Since individuals are replaced during the evolutionary process, it is expected to achieve a performance improvement due to the computational cost reduction associated to the pending terminals optimization of the replaced individuals. In addition to ECO, the application of PBES to avoids the generation of infeasible individuals when terminal symbols represent the components of a real numbered vector that are required to sum a predefined constant.

IV. SOLUTION PROPOSALS AND RESULTS

5. Grammatically Uniform Population Initialization

The Grammatically Uniform Population Initialization (GUPI) is a new population initialization approach that achieves grammatical uniformity. GUPI grants producing any complete derivation of a CFG with the same probability. The grammatical uniformity also helps improving uniformity on solution space. When CFG is unambiguous, GUPI obtains uniformity in both, search and solution spaces. Then, generating any sentence of CFG language is also equiprobable. Uniformity is achieved by dynamically adjusting the probabilities in a stochastic CFG according to the initialization context. The initialization context is defined as the production rules cardinality values during each derivation tree generation within the initialization process. Since production rules cardinalities may vary during derivation tree generation, production rules' probabilities must be dynamically assigned. Additionally, a new code bloat control mechanism is also included. This code bloat control is designed to provide more suitable bounds avoiding initialization bias.

GUPI is designed to initialize grammatically uniform populations independently from the provided CFG. Since (7), the probability of obtaining a complete derivation depends on the probabilities of the involved production rules. Then, these probabilities must be modified to give every complete derivation, $S \overset{*}{\Rightarrow} s$, the same probability to be generated. To this end, for all nonterminal symbol $A \in V$, every terminal string derivation $A \overset{*}{\Rightarrow} s$ must be also generated with the same probability. (8) determines the average probability of any terminal string derivation $\mathrm{A} \overset{*}{\Rightarrow} \mathrm{s}$, applying first a production rule r. Since A nonterminal symbol produces $|A|$ different terminal string derivations, then, $P\left(A \overset{*}{\Rightarrow} s\right) = \frac{1}{|A|}$ must be complied for each terminal string derivation $A \overset{*}{\Rightarrow} s$, independently of the production rule r applied. Therefore, $\langle P\left(A \overset{*}{\Rightarrow} s, r_i : A ::= \gamma_i\right)\rangle = \frac{1}{|A|}$ must be satisfied for every $A \in V$, $r_i \in R$. From (8):

$$\langle P\left(A \overset{*}{\Rightarrow} s, r: A::=\gamma\right)\rangle = \frac{P(r)}{|r|} = \frac{1}{|A|}$$

Then, the production rules probabilities must be assigned as:

$$P(r) = \frac{|r|}{|A|}, \tag{12}$$

which is valid for non-recursive and recursive CFGs. However, code bloat bounds are needed when dealing with recursive CFGs. In this case, the cardinalities of the nonterminal symbols and the production rules may vary during each tree generation as the tree size varies. Therefore, the probabilities of the production rules are dynamically assigned during derivation tree generation according to (12) to comply with grammatical uniformity.

The code bloat control mechanism proposed for GUPI replaces the maximum depth bound by a maximum recursion bound as only recursive CFGs would need code bloat control. The maximum recursion bound reduces code bloat by limiting the number of different recursive derivations. This recursion bound is set at the beginning of the initialization process. During each individual initialization, either recursive or non-recursive derivations can be produced while the maximum recursion bound is not reached. Then, only non-recursive derivations can be generated until the derivation is completed in a finite number of rewriting steps. When several nonterminal symbols on the left-hand side of a production rule can lead to recursive derivations, the remaining recursions (R) must be randomly distributed (R_i) between those nonterminal symbols, so that $\sum R_i = R$. This process avoids generating homogeneous derivation trees and increases diversity.

Table 10 shows the population initialization algorithm. It needs two parameters, the population size (N) and the maximum recursion bound (MR), to produce a grammatically uniform initial population.

Table 10: Grammatically uniform population initialization algorithm.

Repeat N times:

5) *Set axiom as current nonterminal symbol A and $R = MR$*
6) *Obtain the nonterminal A production rules set and compute their probabilities using (12) according to R.*
7) *Choose a production rule $r: A ::= \gamma$ following the computed probabilities.*
8) *If r produces a recursive derivation, set $R = R - 1$*
9) *For every nonterminal symbol $B_i \in \gamma$*
   a) *If B_i do not lead to recursive derivations, then $R_i = 0$*
   b) *Otherwise, set R_i to a random value, such that $\sum R_i = R$.*
   c) *Repeat from step 3 with $A = B_i$ and $R = R_i$*

5.1. Custom Performance Improvements for Grammar Guided Genetic Programming

GUPI is a general-purpose algorithm that produces samples of derivation trees uniformly distributed in the search space defined by a CFG. However, grammatical uniformity may not always provide the best initial population distribution in terms of evolutionary performance since the CFG definition is completely dependent on the problem to solve. Under some circumstances, it can be beneficial altering grammatical uniformity to improve the evolutionary performance. There are two main scenarios where modifications may be positive: CFGs that produce structurally similar derivation trees with many different terminal symbol combinations (leaves); and CFGs with recursive derivations that cause main changes in phenotype.

If the cardinality of CFG terminal symbol set is high, then, there are usually many similar derivation trees with different combinations of terminal symbols in their leaves. In this scenario, GUPI likely tends to generate them as they represent a big subset of the search space. Population initialization is then biased to those derivation trees, producing a poorly diverse population. This effect is intensified when recursive derivations are involved. This issue can be solved by excluding terminal symbols from the production rules probabilities calculation during derivation tree generation. This way grammatical uniformity does not depend on different combinations of terminal symbols and the population distribution bias will be reduced improving the global evolutionary performance. This solution is

implemented as following: when $A \in V_o$ we take $|A| = 1$, and for $A \in V_i$ with $i \neq 0$, $|A|$ is given by (6).

In the second scenario, GUPI likely produces deep derivation trees as recursive production rules cardinalities are usually very high. Recursive production rules may cause substantial changes in individual's phenotype. In this case, GGGP performance can be improved by altering grammatical uniformity to pursue both: recursion uniformity in the first place, and then, grammatical uniformity. Recursion uniformity seeks to produce derivation trees with a uniform amount of recursive derivations. To this end, a ramped-like approach is adopted: every time a derivation tree is created, a recursion bound is randomly set between 0 and a predefined maximum recursion bound.

5.2. Results

With the aim of showing the advantages and main features of the GUPI, a diverse set of experiments using CFGs representing different search spaces is presented. In addition to $G_{Recursive}$ (9), five new CFGs are presented to reveal the limitations and goodness of population initialization methods. These CFGs encode the search space of different real world problems and intelligent system definitions. The CFGs are:

- A CFG designed to generate feed-forward neural networks with one hidden layer and a variable number of hidden neurons (Couchet, et al., 2007): G_{FFNN} (13). One recursive production rule is defined to determine the number of neurons in the hidden layer. Moreover, this CFG provides a mechanism to define the connexion between the neurons of the artificial neural network. In particular, the employed CFG generates feed-forward neural networks with 4 inputs and 1 output.

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

$V = \{ILS, DLS, H, IL, OL, ?\}$

$\Sigma = \{0, 1\}$

$R = \{$

r_1	$S ::= DLS\ ILS$
r_2	$S ::= DLS$
r_3	$ILS ::= H\ ILS$
r_4	$ILS ::= H$
r_5	$H ::= IL\ OL$
r_6	$IL ::= 1\ ?\ ?\ ?$
r_7	$IL ::= ?\ 1\ ?\ ?$
r_8	$IL ::= ?\ ?\ 1\ ?$
r_9	$IL ::= ?\ ?\ ?\ 1$
r_{10}	$OL ::= 1$
r_{11}	$OL ::= ?\ ?\ ?\ ?$
r_{12}	$? ::= 1$
r_{13}	$? ::= 0$

$\}$ (13)

- A CFG employed to generate fully connected deep feed-forward neural networks that contains several hidden layers: G_{DFFNN} (14). This CFG contains two recursive production rules: one to determine the number of hidden layers and another one to determine the number of neurons per hidden layer.

$G_{DFFNN} = (V, \Sigma, R, L)$

$V = \{L, H\}$

$\Sigma = \{0, 1\}$

$R = \{$

r_1	$L ::= H\ 0\ L$
r_2	$L ::= H$
r_3	$H ::= 1\ H$
r_4	$H ::= 1$

$\}$ (14)

- A CFG designed to obtain knowledge bases for rule based systems: G_{RBS} (15). This CFG defines two recursive production rules: r_2 and r_5. r_2 determines the rules included in the knowledge base and r_5 the clauses of each rule.

G_{RBS}= (V, Σ, R, S)

V = {RBS, RULE, ANTECEDENT, CONSEQUENT, EVIDENCE, CLAUSE, VALUE, EQUALITY OPERATOR}

Σ = {if, then, =, !=, &, ,| V1, V2, V3, V4, V5, T, F}

R = {

r_1 S ::= RBS

r_2 RBS ::= RULE RBS

r_3 RBS ::= RULE

r_4 RULE ::= if ANTECEDENT then CONSEQUENT

r_5 ANTECEDENT ::= EVIDENCE OPERATOR ANTECEDENT

r_6 ANTECEDENT ::= EVIDENCE

r_7 EVIDENCE ::= CLAUSE EQUALITY VALUE

r_8 EQUALITY ::= = (15)

r_9 EQUALITY ::= !=

r_{10} OPERATOR ::= &

r_{11} OPERATOR ::= |

r_{12} CONSEQUENT ::= V1

r_{13} CONSEQUENT ::= V2

r_{14} CONSEQUENT ::= V3

r_{15} CONSEQUENT ::= V4

r_{16} CLAUSE ::= V1

r_{17} CLAUSE ::= V2

r_{18} CLAUSE ::= V3

r_{19} CLAUSE ::= V4

r_{20} VALUE ::= T

r_{21} VALUE ::= F

}

- A CFG designed to define the expressions of a symbolic regression optimization problem: G_{SR} (16). This CFG contains two recursive production rules that include new operations to new expressions: r_2 and r_3. r_2 creates unary operations, while r_3 binary operations. Note r_3 produces a binary recursion since two new expressions are produced from one.

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

$V = \{EXP, OP\_1, OP\_2\}$

$\Sigma = \{(,), +, -, /, *, ^, sin, cos, log, V\}$

$R = \{$

r_1	$S ::= EXP$
r_2	$EXP ::= (OP\_1\ EXP)$
r_3	$EXP ::= (OP\_2\ EXP\ EXP)$
r_4	$EXP ::= V$
r_5	$OP\_1 ::= ^$
r_6	$OP\_1 ::= log$
r_7	$OP\_1 ::= sin$
r_8	$OP\_1 ::= cos$
r_9	$OP\_2 ::= +$
r_{10}	$OP\_2 ::= -$
r_{11}	$OP\_2 ::= /$
r_{11}	$OP\_2 ::= *$

$\}$ (16)

- A CFG designed to obtain Boolean functions that solve problems such as parity (White, et al., 2013): G_{BP} (17). As G_{SR}, it contains two recursive production rules that include new operations to new expressions: r_2 and r_3. r_2 creates unary operations, while r_3 binary operations. However, in this case the cardinality of terminal symbol set is more reduced since only to possible values $(1, 0)$ can be used in the operations.

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

$V = \{EXP, OP, b\}$

$\Sigma = \{and, or, not, ^, 1, 0, (,)\}$

$R = \{$

r_1	$S ::= EXP$
r_2	$EXP ::= (not\ EXP)$
r_3	$EXP ::= (EXP\ OP\ EXP)$
r_4	$EXP ::= b$
r_5	$OP ::= and$
r_6	$OP ::= or$
r_7	$OP ::= ^$
r_8	$OP ::= 0$
r_9	$OP ::= 1$

$\}$ (17)

Two different experiments have been performed for each of these CFGs to show the goodness of GUPI: initialization and optimization experiments. Initialization experiments are presented in the subsection 5.2.1, and optimization experiments in subsection 5.2.2.

5.2.1. Population initialization experiments

The initialization experiments are designed to show the distribution of the initial population when dealing with all the presented CFG. Since all the CFG possess recursive production rules that cause substantial changes in individual's phenotype, recursion uniformity is firstly applied. 1000 individuals are generated in every experiment to show the population distribution. The number of recursion derivations of the initial population individuals is calculated for GUPI and the regular GGGP population initialization method (Table 8) to show their capabilities to obtain representative samples. These results are represented in histograms where the abscissa axis represent the number of recursive derivations per derivation tree and the ordinate axis the absolute frequency of derivation trees for each abscissa value.

Custom Recursive Grammar Population Initialization

When using the $G_{Recursive}$ (9), the regular GGGP population initialization already shows a bias towards derivation trees with few recursive derivations when recursion bound is set to 5 (Figure 21). Conversely, the GUPI approximates a uniform distribution of recursive derivations

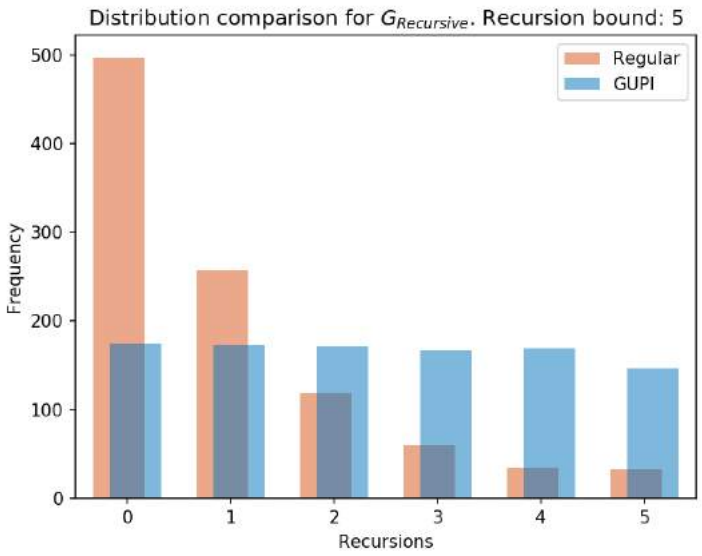

Figure 21: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 5 for the $G_{Recursive}$.

The recursion distribution for the GUPI and the regular approach have similar shapes for higher recursion bounds. It is noteworthy that the regular approach shows a soft-bound on 7 recursive derivations and almost no derivation tree surpasses that limit. This phenomenon can be already observed when the user recursion bound is set to 10 (Figure 22). On the contrary, GUPI still approximates a uniform distribution independently of the recursion bound. Figure 22 to Figure 25 show the initial population distribution for the recursion bounds 10, 20, 50, and 100.

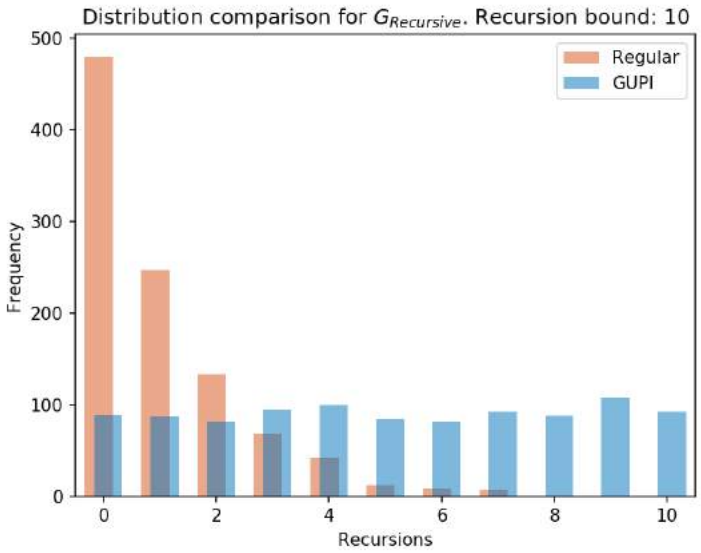

Figure 22: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 10 for the $G_{Recursive}$.

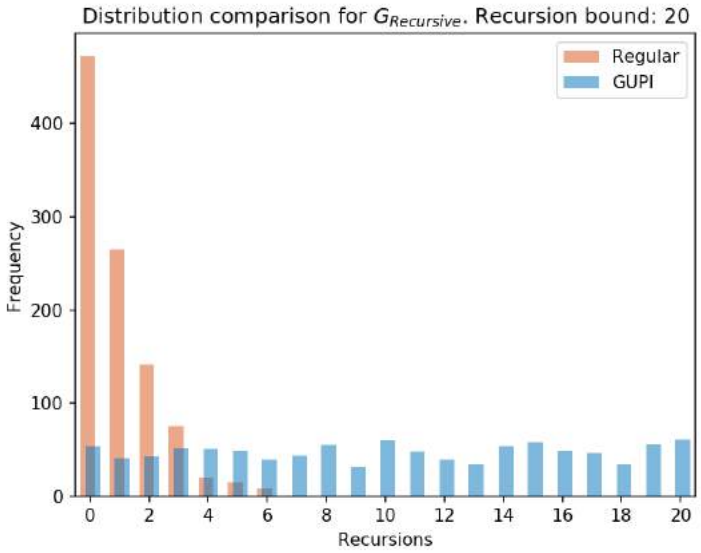

Figure 23: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 20 for the $G_{Recursive}$.

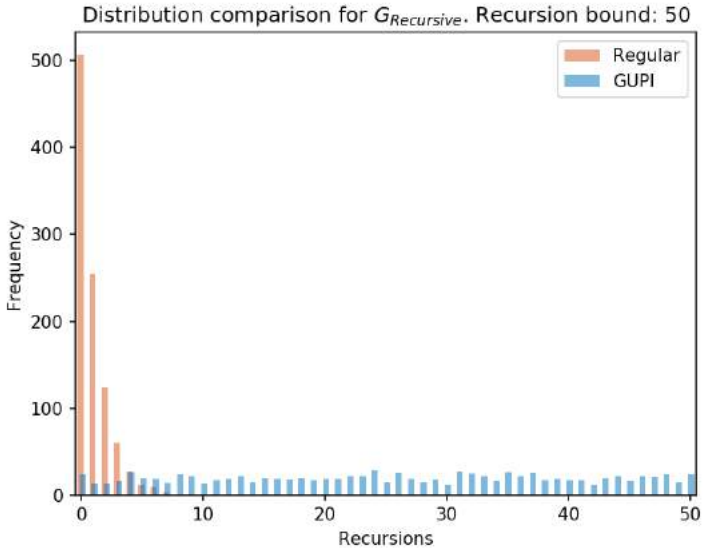

Figure 24: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 50 for the $G_{Recursive}$.

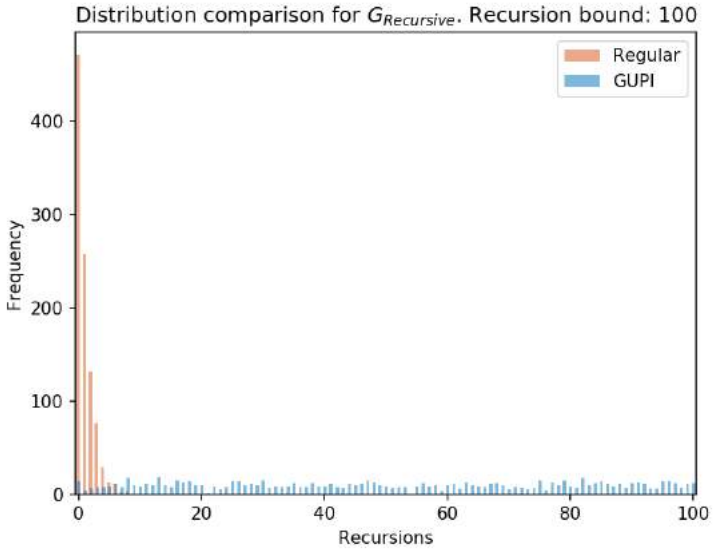

Figure 25: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 100 for the $G_{Recursive}$.

Feed-forward Neural Network (1 hidden layer) Population Initialization

Similar results are obtained for the G_{FFNN} (13) since it only possesses one recursive production rule like $G_{Recursive}$ does. The regular GGGP population initialization shows a bias towards derivation trees with few recursive derivations and it shows a soft-bound on 7 recursive derivations that almost no derivation tree surpasses.

Conversely, the GUPI adequately approximates a uniform distribution of recursive derivations. Figure 26 to Figure 30 show the initial population distribution for the recursion bounds 5, 10, 20, 50, and 100.

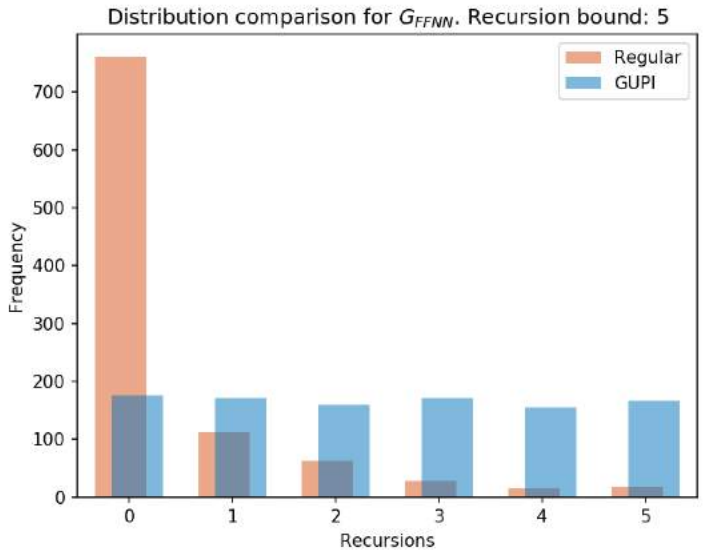

Figure 26: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 5 for the G_{FFNN}.

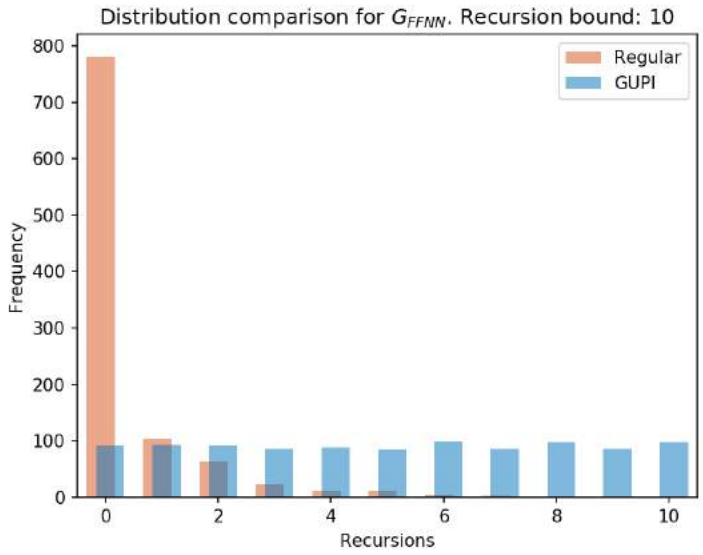

Figure 27: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 10 for the G_{FFNN}.

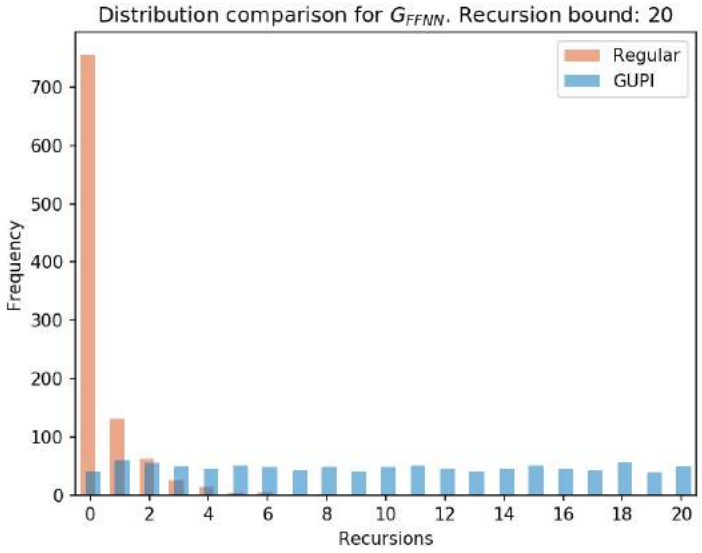

Figure 28: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 20 for the G_{FFNN}.

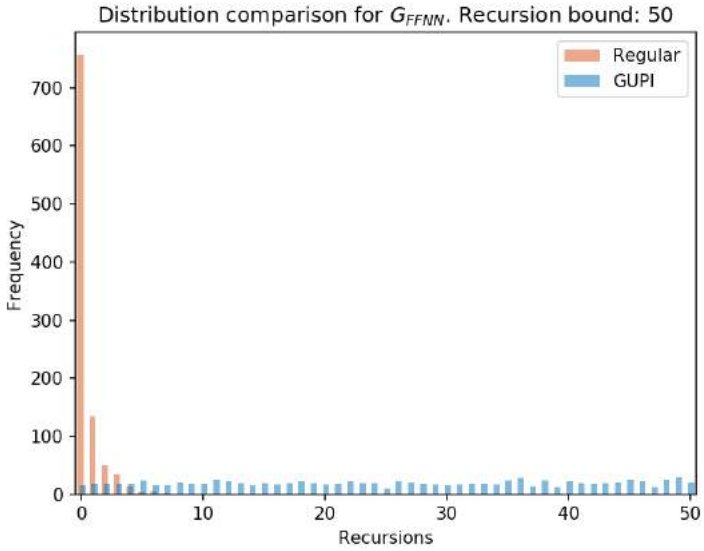

Figure 29: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 50 for the G_{FFNN}.

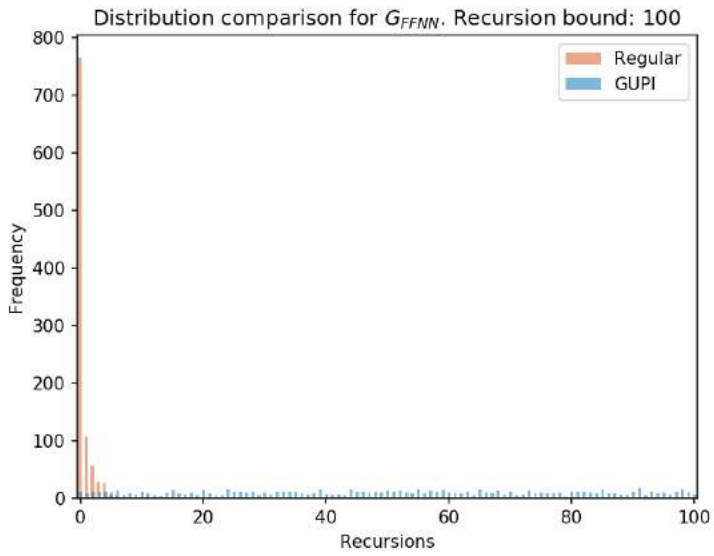

Figure 30: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 100 for the G_{FFNN}.

Deep Feed-forward Neural Network Population Initialization

The population initialization with the G_{DFFNN} (14) shows little differences compared to the two previous ones since it contains two nested recursive production rules (14). In this case, the regular population initialization approach reduces its bias towards derivation trees with few recursive derivations. However, it is still biased and, in fact, it still shows a soft bound, this time, close to 13 recursive derivations (Figure 33). On the contrary, the GUPI still approximates a uniform distribution of recursive derivations. The recursion bounds have been increased since the G_{DFFNN} is used to produce large neural networks with several layers and many hidden neurons. Therefore, many recursive derivations are usually applied. Figure 31 to Figure 36 show the initial population distribution for the recursion bounds 5, 10, 20, 100, and 250. Experiments have been also performed for the recursive derivations bound 500. The regular and GUPI approaches shows similar results. However, the histogram for this bound cannot be shown since the bars are not clear.

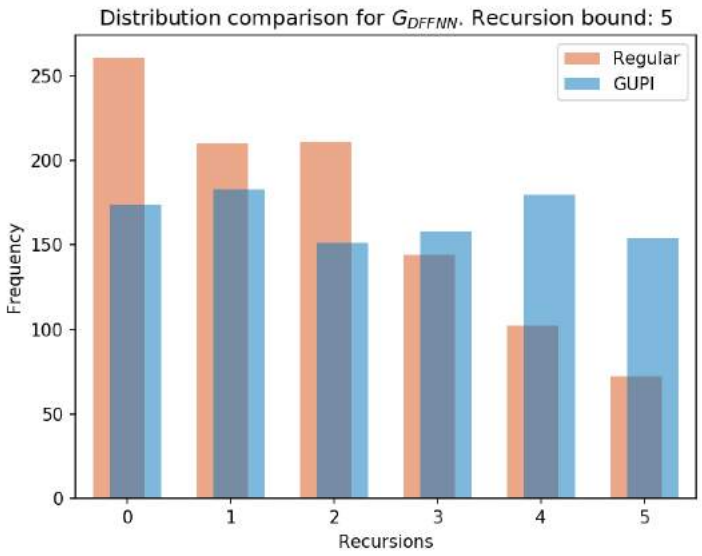

Figure 31: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 5 for the G_{DFFNN}.

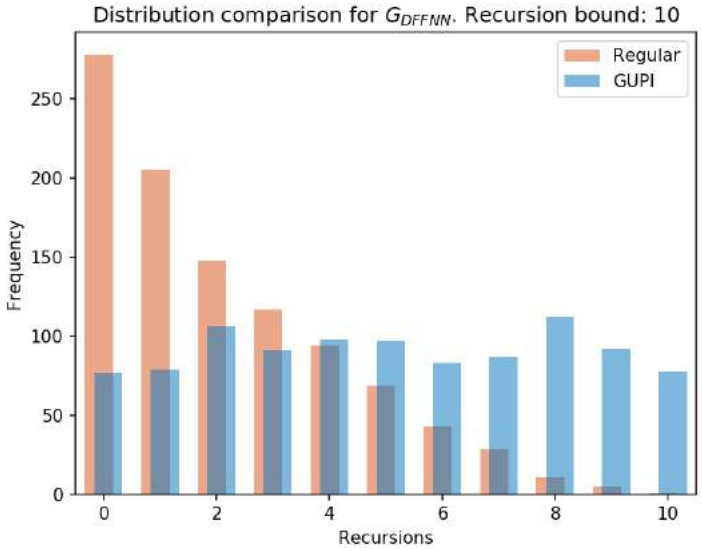

Figure 32: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 10 for the G_{DFFNN}.

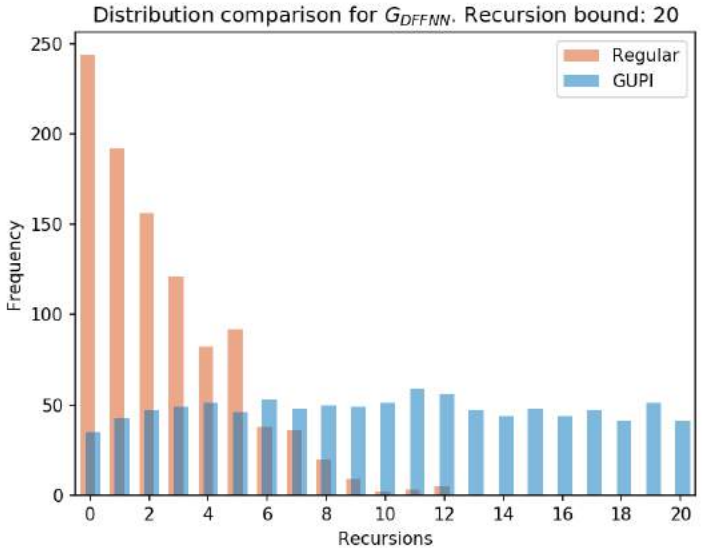

Figure 33: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 20 for the G_{DFFNN}.

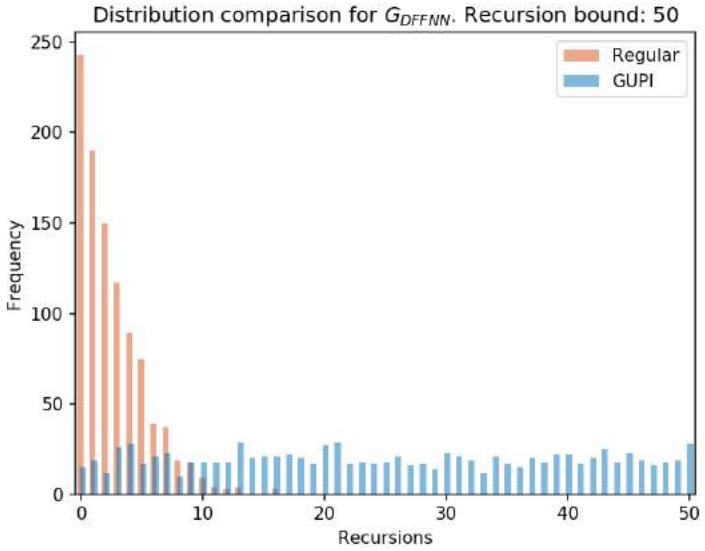

Figure 34: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 50 for the G_{DFFNN}.

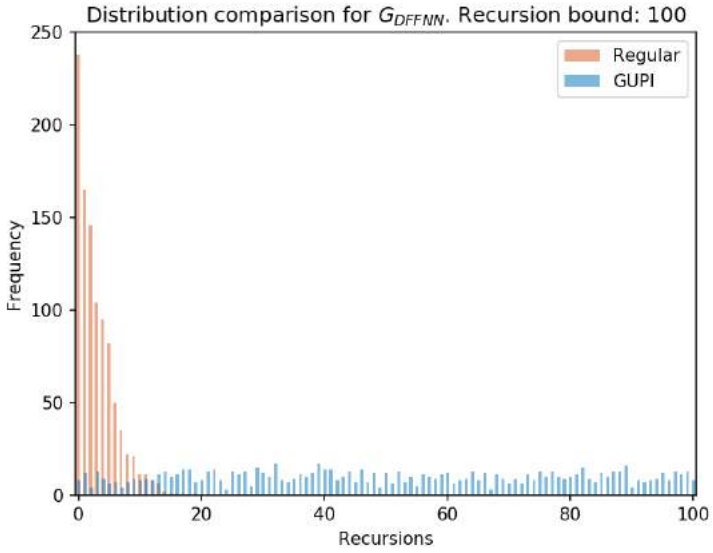

Figure 35: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 100 for the G_{DFFNN}.

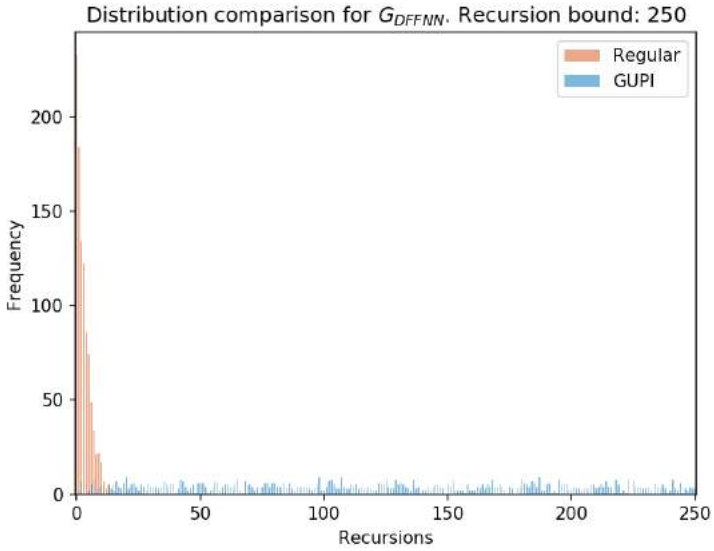

Figure 36: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 250 for the G_{DFFNN}.

Rule Based System Population Initialization

Similar results to the G_{DFFNN} experiments are obtained for the G_{RBS} since it also possesses two nested recursive production rules (15). The regular GGGP population initialization shows a bias towards derivation trees with few recursive derivations and it shows again a soft-bound on 13 recursive derivations (Figure 39) that almost

no derivation tree surpasses. Conversely, the GUPI adequately approximates a uniform distribution of recursive derivations. Figure 37 to Figure 41 show the initial population distribution for recursion bounds 5, 10, 20, 50, and 100.

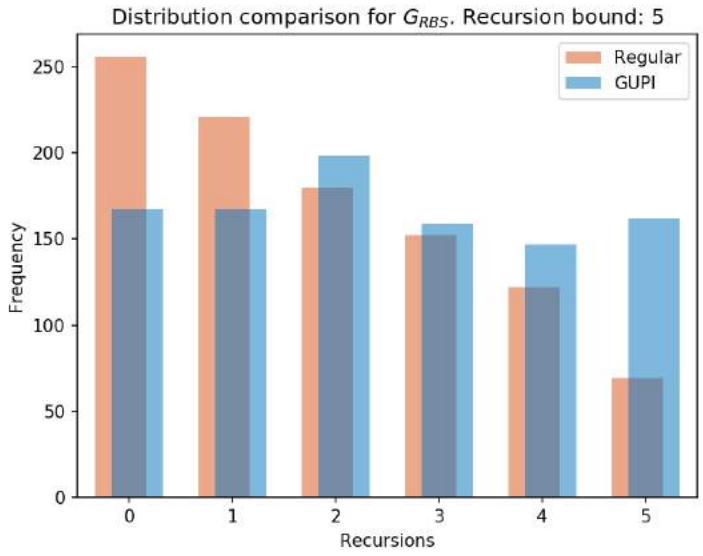

Figure 37: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 5 for the G_{RBS}.

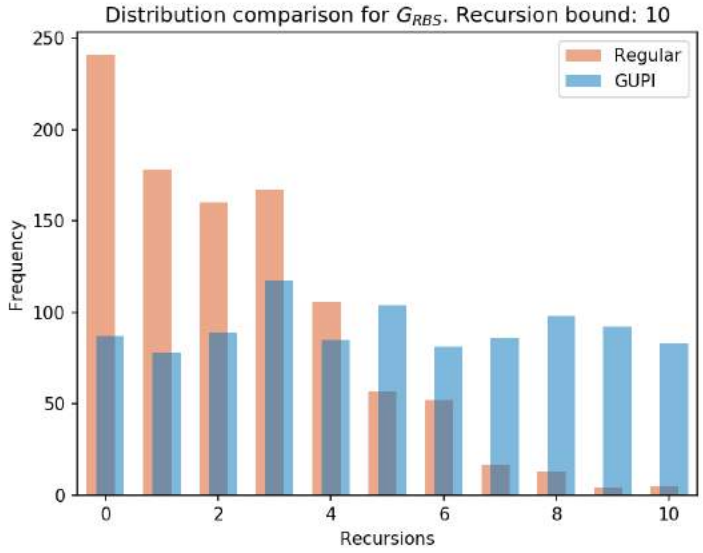

Figure 38: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 10 for the G_{RBS}.

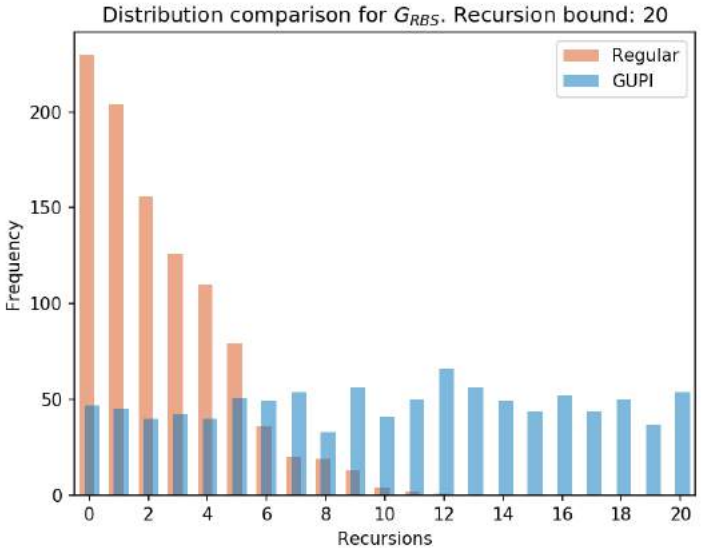

Figure 39: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 20 for the G_{RBS}.

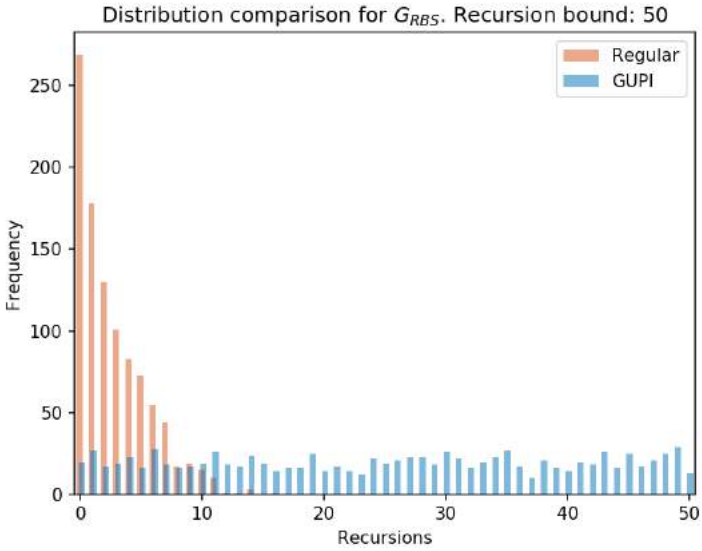

Figure 40: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 50 for the G_{RBS}.

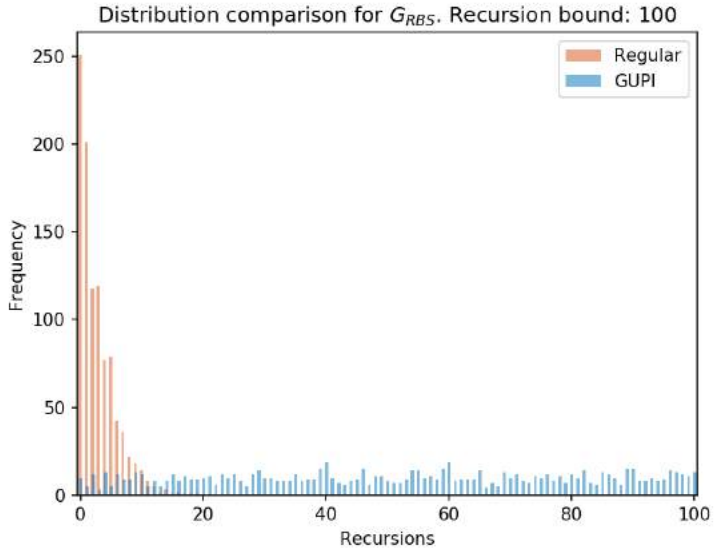

Figure 41: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 100 for the G_{RBS}.

Symbolic Regression Population Initialization

The G_{SR} has two different recursive production rule (16) that lead the regular population initialization approach to produce derivation trees with more recursive derivations. In particular, r_3 facilitates the selection of recursive production rules since every time it is selected the probability of selecting recursive production rules is doubled. In this case, from 5 to almost 20 recursion bounds, the regular population initialization is able to produce derivation trees of any number of recursive derivations (Figure 42 to Figure 44). Nevertheless, there is still a bias towards derivation trees with few recursive derivations. Moreover, when the recursion bound is increased, soft bound appears again and derivation trees do not usually surpass 30 recursive derivations (Figure 45). Contrariwise, the GUPI still adequately approximates a uniform distribution of recursive derivations independently of the recursion bound defined. Figure 42 to Figure 46 show the initial population distribution for recursion bounds 5, 10, 20, 50, and 100.

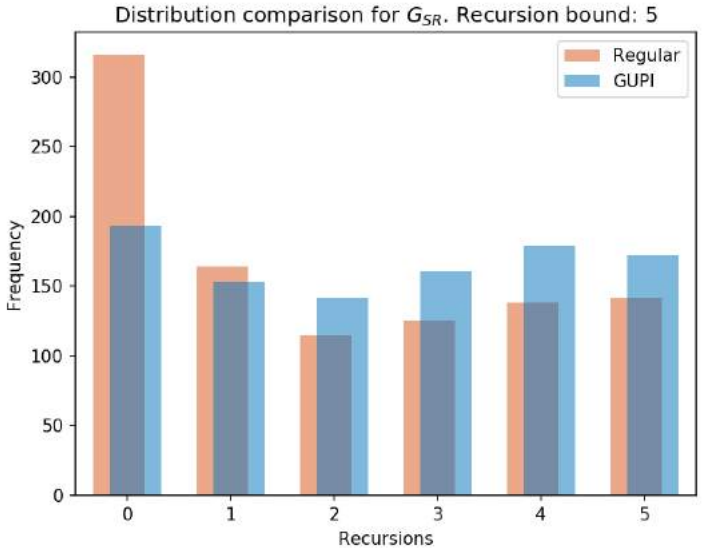

Figure 42: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 5 for the G_{SR}.

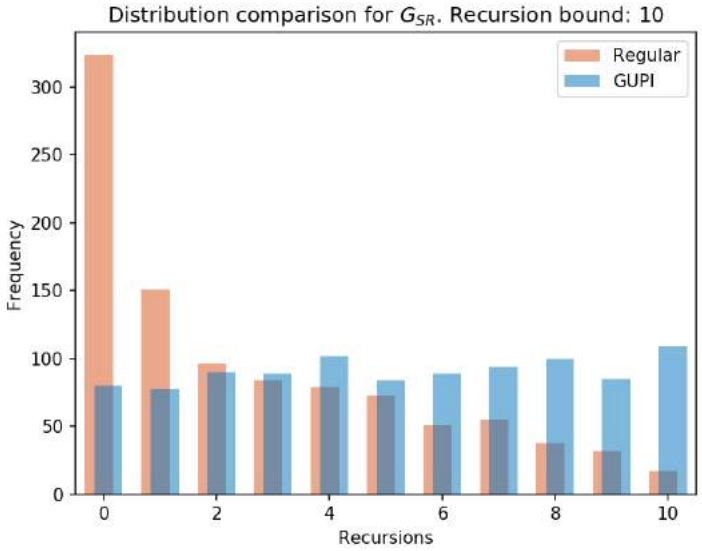

Figure 43: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 10 for the G_{SR}.

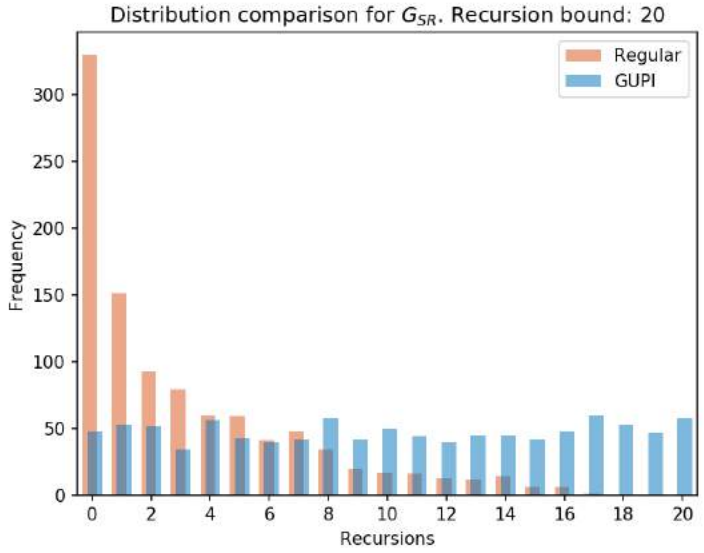

Figure 44: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 20 for the G_{SR}.

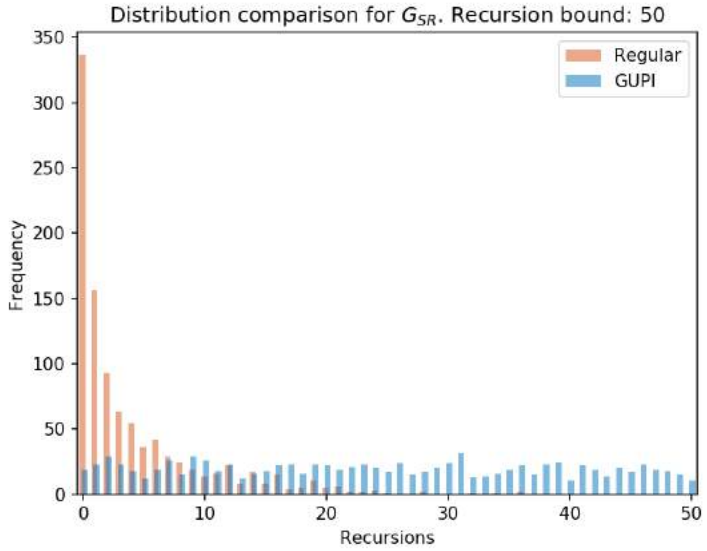

Figure 45: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 50 for the G_{SR}.

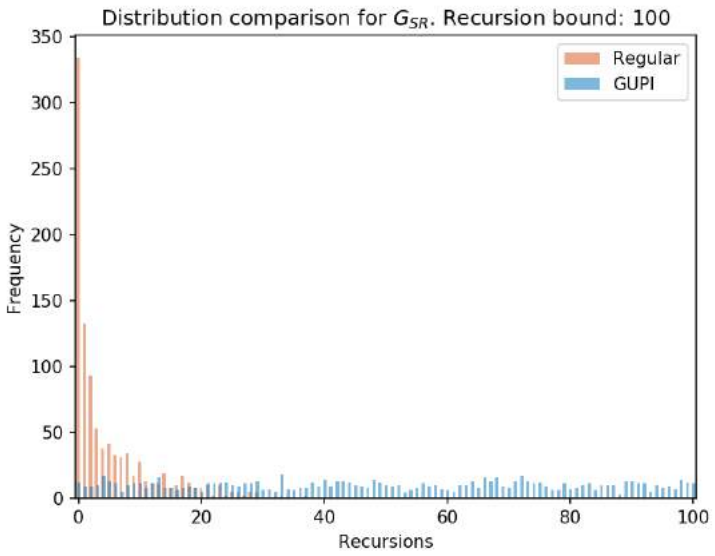

Figure 46: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 100 for the G_{SR}.

Boolean Function Problems Population Initialization

Similar results to the G_{SR} are obtained for the G_{BP} since it also has two recursive production rules with the same characteristics (17). The regular GGGP population initialization shows a bias towards derivation trees with few recursive derivations and it shows a soft-bound close to 30 recursive derivations that almost no derivation tree surpasses. Conversely, the GUPI adequately approximates a uniform distribution of recursive derivations independently to the recursion bound defined. Figure 47 to Figure 51 show the initial population distribution for recursion bounds 5, 10, 20, 50, and 100.

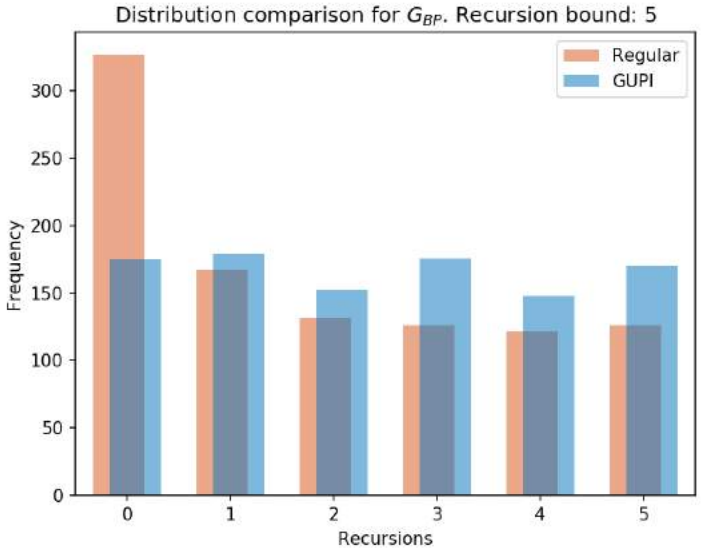

Figure 47: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 5 for the G_{BP}.

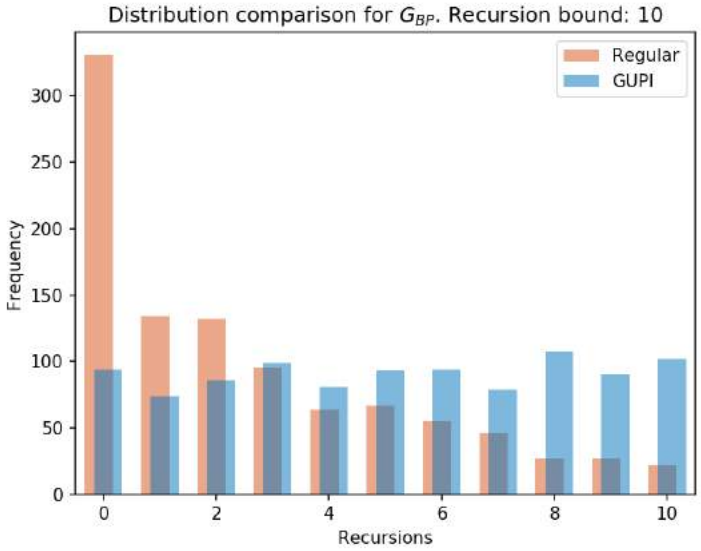

Figure 48: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 10 for the G_{BP}.

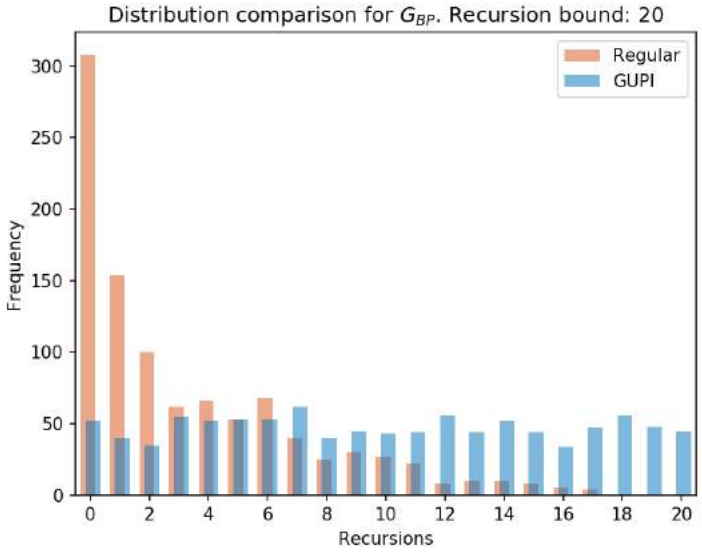

Figure 49: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 20 for the G_{BP}.

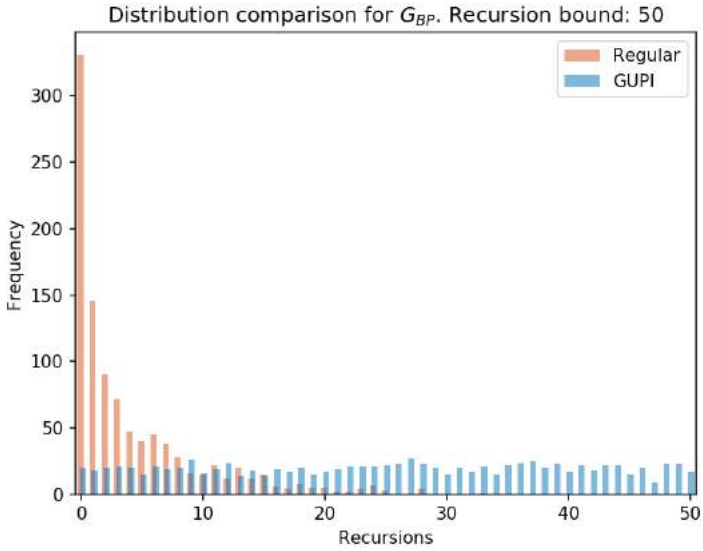

Figure 50: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 50 for the G_{BP}.

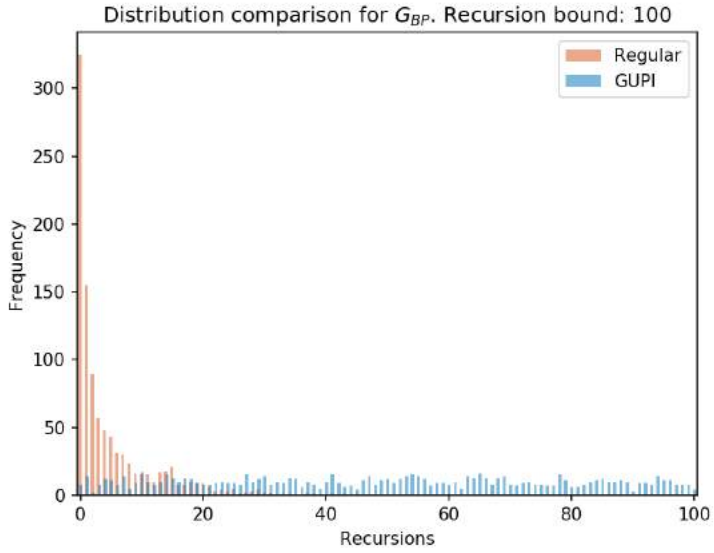

Figure 51: Population initialization recursions comparison of GUPI and a regular population initialization with a maximum recursion bound of 100 for the G_{BP}.

5.2.2. Optimization experiments

The optimization experiments are designed to compare the performance of the overall GGGP evolutionary process when using either GUPI or the regular GGGP population initialization (Table 8). Different experiments are conducted to show the performance of both approaches when trying to obtain derivation trees with different number of recursive derivations: 5, 10, 20, 50, 100 and 250 recursive derivations. For each of these values, a random derivation tree with that number of recursive derivations is generated. These are the target derivation trees that the GGGP optimization problem must find. Then, an experiment is performed for each of the target derivation trees where each GGGP approach tries to obtain that derivation tree. Derivation trees with 0 recursive derivations are not included in the experiments since the initial population usually contains the target derivation tree. The GGGP fitness function is the hamming distance between the word encoded in the target derivation tree and the word of the evaluated derivation tree. The derivations words are compared, so that, for each position of the words, the fitness value is increased by one every time the elements in that position are different. If the lengths of the words are not the same, the difference of the lengths

is summed to the fitness value since the last elements of the longest word cannot be compared. The evolutionary process seeks to minimize the fitness value. The evolutionary process converges if it finds the target derivation tree or it accomplishes 50 generations without improving the average population fitness. The WX and an incremental EDA are employed as variation operators together with both population initialization approaches to compare the performance when using two of the most common GGGP optimization approaches. WX chooses 2 individuals to produce 2 new individuals. EDA chooses 50% of the population to produce a new whole population. The EDA's incremental learning rate is set to 0.3 so that a 0.7 weight is applied to the previous probabilistic model and 0.3 weight to the selected individuals probabilistic model. Both approaches employ the tournament selection to improve the population diversity. No mutation or immigration are applied to reduce the random component of the evolutionary process and facilitate the comparison. Population size is fixed to 100. 50 runs are performed for each experiment with the same target derivation. The number of generations and the fitness are measured to the determine the advantages of generating more representative initial populations.

Two types of plots are employed to show the results: an evolution of the average fitness of each GGGP experiment run and a statistical study of the best fitness obtained and the generations when GGGP converged for all runs of each experiment. In the evolution chart, the abscissa axis represents the GGGP generation when the average population fitness value, represented in ordinate axis, is collected. The fitness evolution is represented as a line that starts at the beginning of the GGGP optimization process (generation 0) and finishes when GGGP converges. There is an evolution line for each of the 50 execution runs of each approach. There are two statistical studies, one for best fitness obtained when GGGP converges and another for the number of generation when GGGP converges. Each statistical study comprises two charts. Both share the abscissa axis that represents the experiment (in number of recursive derivations of the target derivation tree searched). The lower chart shows some descriptive statistics of each

approach and experiment by mean of boxplots. The ordinate represents the random variable studied (the best fitness obtained in the resulting population or the total number of generations). The upper chart shows a one-way analysis of variance (one way ANOVA) with a significance level $\alpha = 0.05$ for each experiment and pair of approaches. In this particular case, an ANOVA for each experiment. An analysis of variance is conducted to gather empirical evidence of whether the differences between the means of the random variables are statistically significant for each approach. One of the required conditions of an ANOVA is that the variances of the groups should be equivalent. However, ANOVA is robust to this violation when the groups are of equal or near equal size. This condition is satisfied as 50 samples are obtained for each approach and experiment. The ANOVA independent variable is the approach version. The dependent variable is the best population fitness or the number of generations when GGGP converges. The p-value, represented in the ordinate axis, shows the result of the ANOVA tests, proving that the null hypothesis, H_0, (each approach random variable means are equal) can be rejected if the resulting p-value is below the significance level $\alpha = 0.05$, that is, $p < 0.05$. Thus, statistical evidence is provided to state if there is statistically significant difference between the approaches compared. A dashed line in $p = 0.05$ is provided to facilitate the comparison of the obtained p-value and the fixed significance level value.

Feed-forward Neural Network (1 hidden layer) Optimization

Whigham Crossover

The performed experiments with the G_{FFNN} shows the population initialization barely affects the evolutionary process when the searched derivation tree has less than 20 recursive derivations (Figure 52 and Figure 53). For those cases, both population initialization approaches produce a suitable representative sample of the population. Therefore, there is no statistically significant difference between the average fitness of the individuals obtained (Figure 58). When the target derivation tree has 20 or more recursive derivations (Figure 54 to Figure 57), the fitness mean of the best individuals obtained by the evolutionary process initialized with GUPI

outscores the fitness mean of the evolutionary process using the regular population initialization (Figure 58). Figure 54 shows the first evidences of the difficulties of the evolutionary process when using the regular population initialization. The fitness evolution slope is lower than the previous experiments (Figure 52 and Figure 53) where the target derivation trees have less than 20 recursive derivations. This happens due to limitations of the regular population initialization to provide representative samples of the population. Moreover, Figure 55 and Figure 56 show the premature convergence of the regular population initialization approach due to these same difficulties. Statistical evidences are provided to state there is a statistically significant difference between the fitness means when the target derivation tree has 20 or more recursive derivations (Figure 58). Moreover, Figure 59 shows how GUPI reduces the average iterations needed to obtain an optimal solution independently of the number of recursions of the target derivation tree.

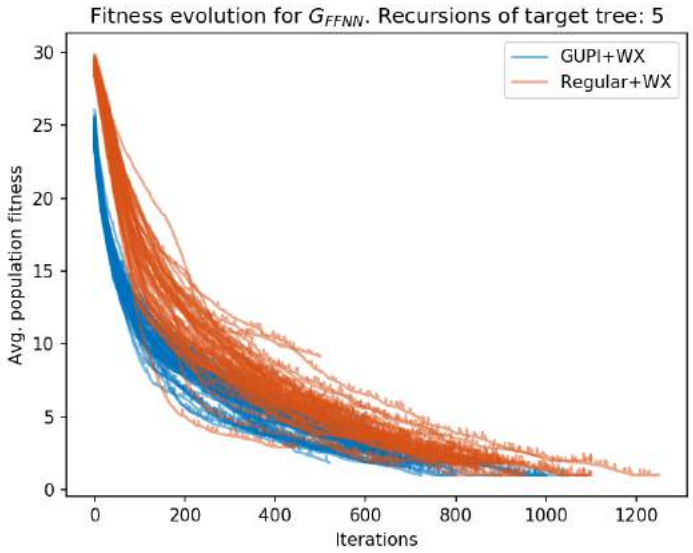

Figure 52: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{FFNN} with 5 recursions is searched. 50 executions run.

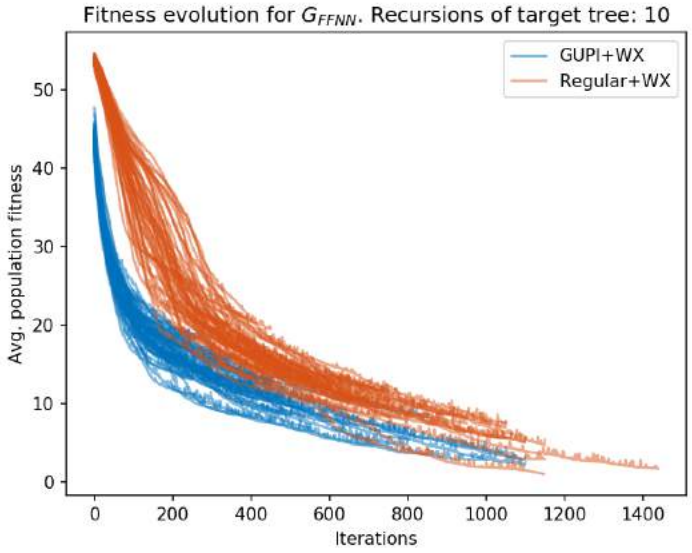

Figure 53: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{FFNN} with 10 recursions is searched. 50 executions run.

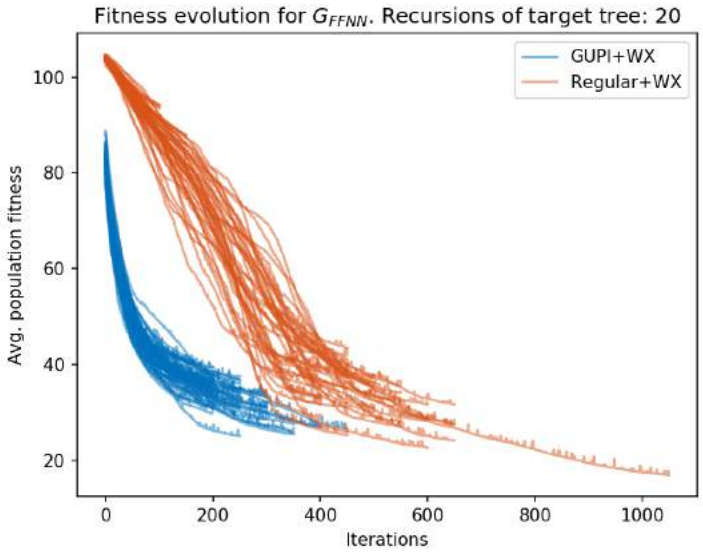

Figure 54: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{FFNN} with 20 recursions is searched. 50 executions run.

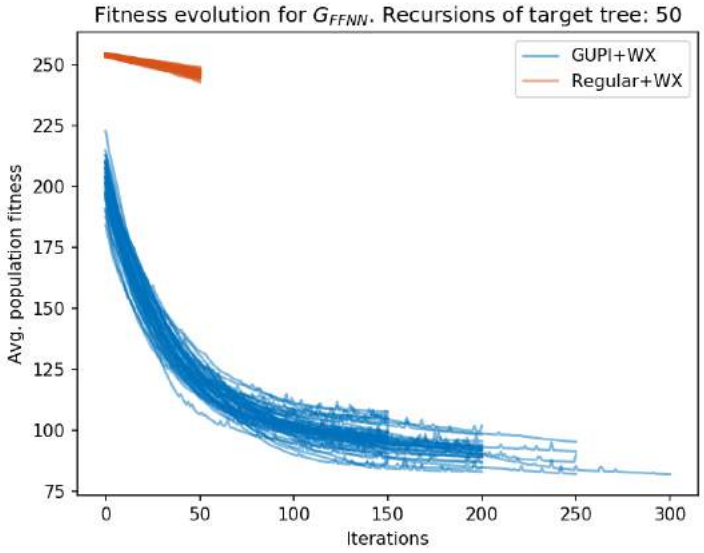

Figure 55: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{FFNN} with 50 recursions is searched. 50 executions run.

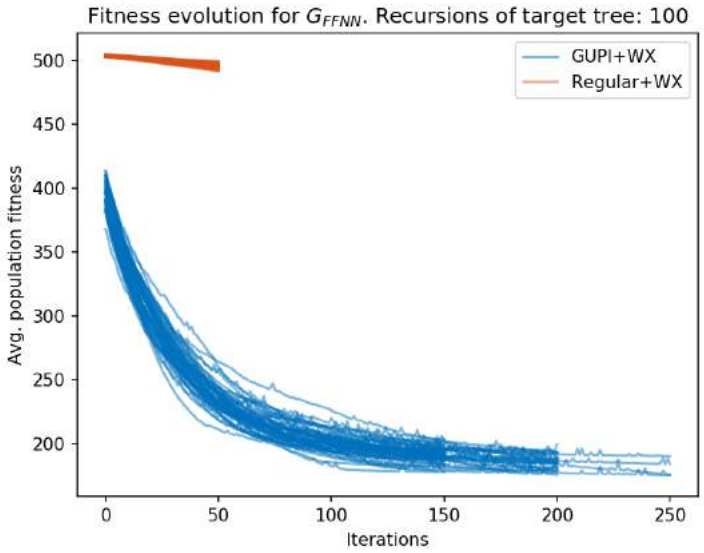

Figure 56: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree ofG_{FFNN} with 100 recursions is searched. 50 executions run.

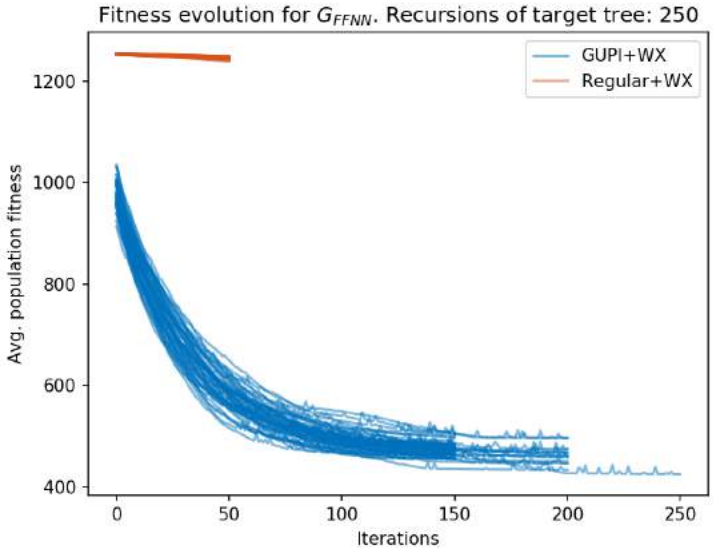

Figure 57: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{FFNN} with 250 recursions is searched. 50 executions run.

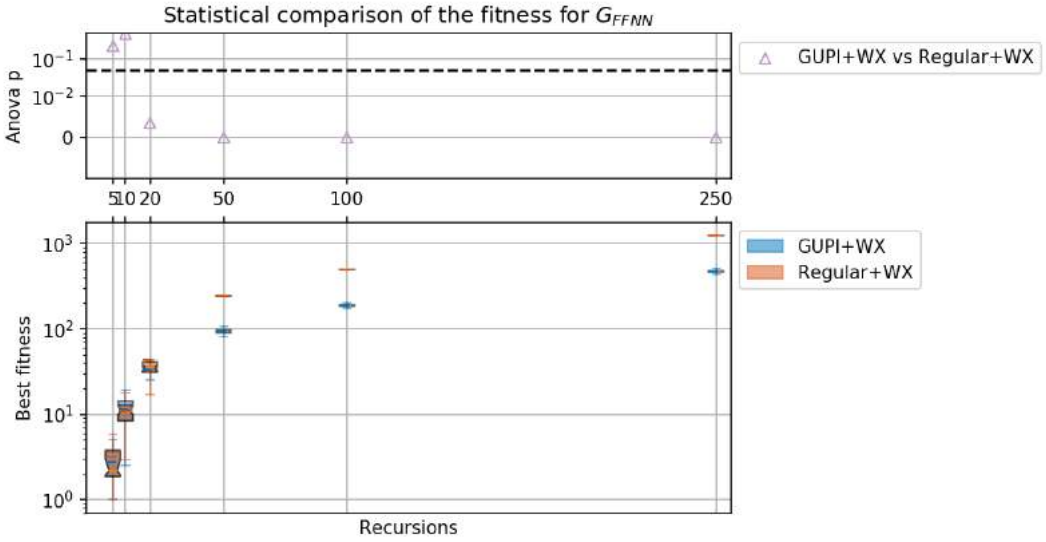

Figure 58: One-way ANOVA of the average fitness for G_{FFNN} to compare GUPI with a regular population initialization when using a Whigham crossover. 50 executions run per target derivation tree.

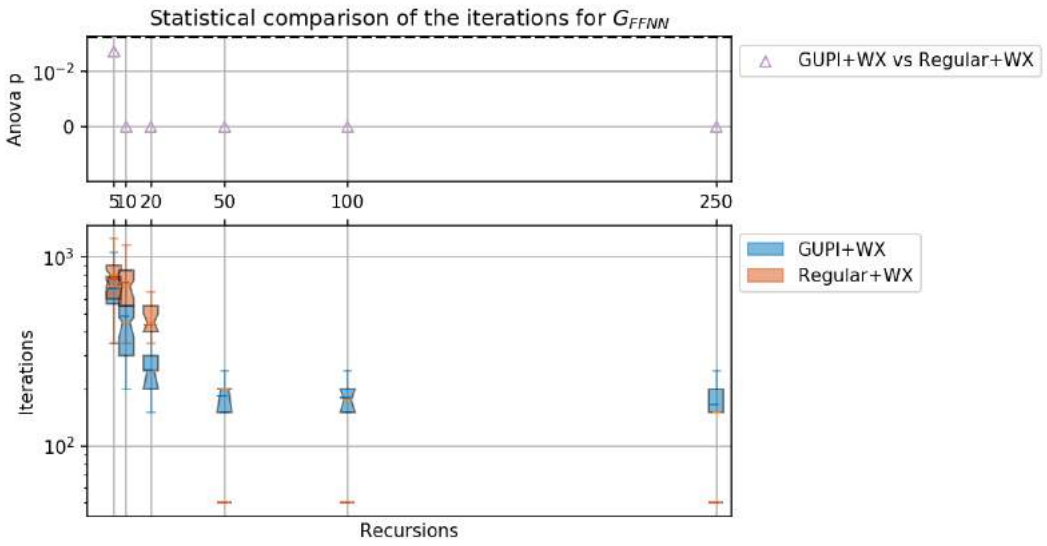

Figure 59: One-way ANOVA of the average iterations for G_{FFNN} to compare GUPI with a regular population initialization when using a Whigham crossover. 50 executions run per target derivation tree.

Estimation of Distribution Approach

The performed experiments with the G_{FFNN} in a GGGP-EDA approach shows GUPI significantly improves the evolutionary process in terms of fitness independently of the target derivation tree (Figure 66). Figure 60 to Figure 64 shows the average fitness of the GUPI approach is already significantly lower during the whole evolutionary process, and, in general terms, the approach using the regular population initialization prematurely converges (Figure 67). Figure 66 provides statistical evidences to state there is a statistically significant difference between the fitness means.

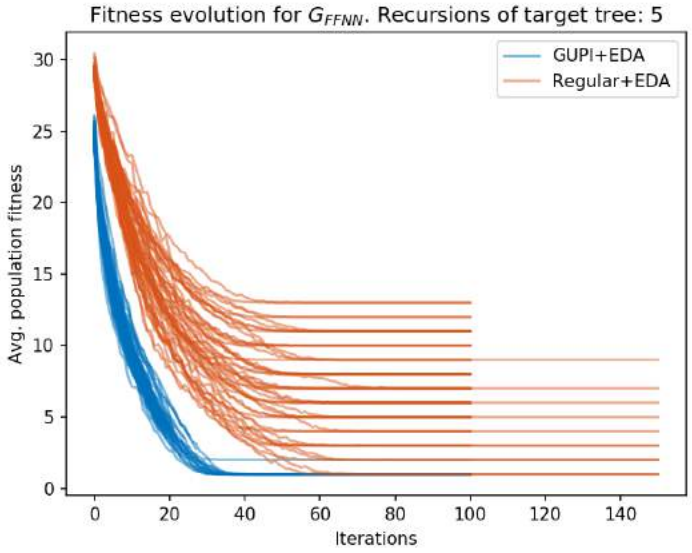

Figure 60: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{FFNN} CFG with 5 recursions is searched. 50 executions run.

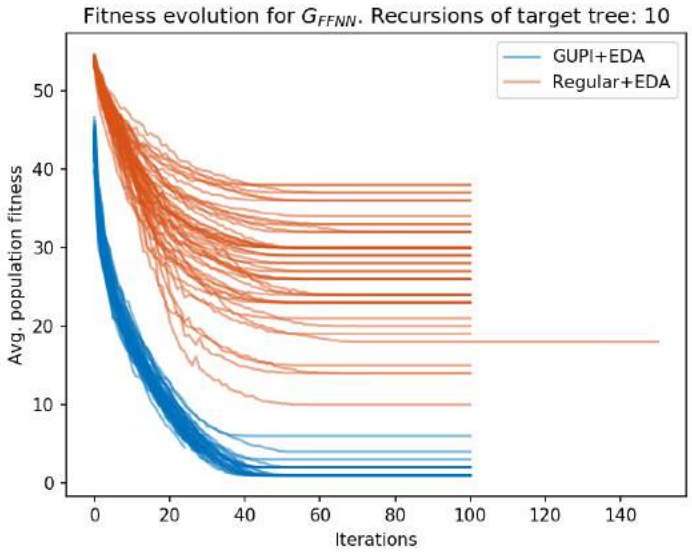

Figure 61: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{FFNN} CFG with 10 recursions is searched. 50 executions run.

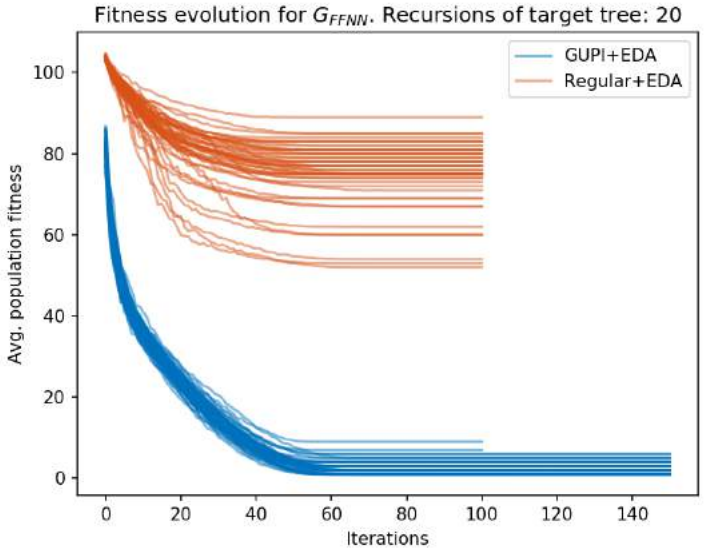

Figure 62: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{FFNN} CFG with 20 recursions is searched. 50 executions run.

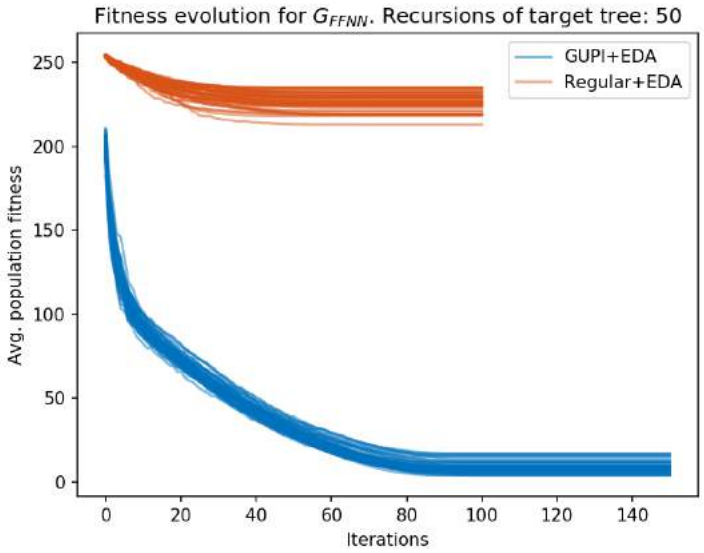

Figure 63: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{FFNN} CFG with 50 recursions is searched. 50 executions run.

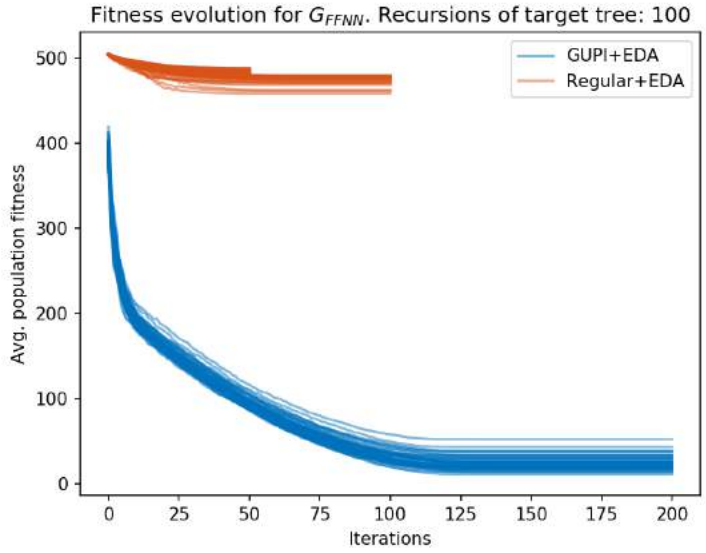

Figure 64: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{FFNN} with 100 recursions is searched. 50 executions run.

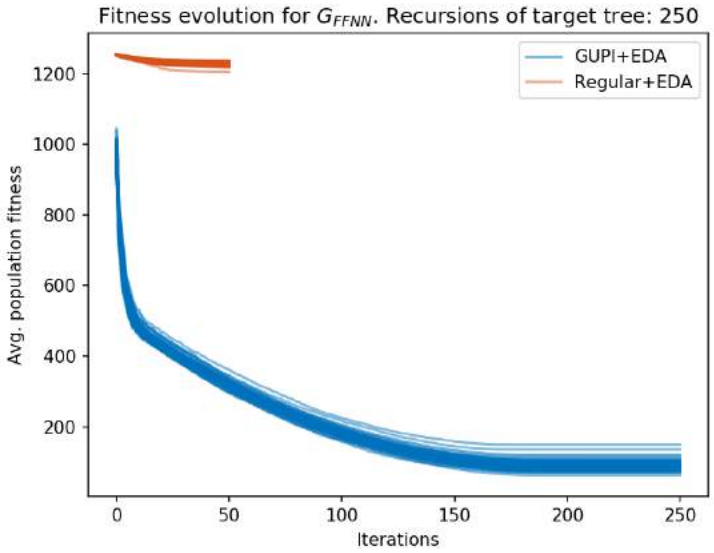

Figure 65: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{FFNN} with 250 recursions is searched. 50 executions run.

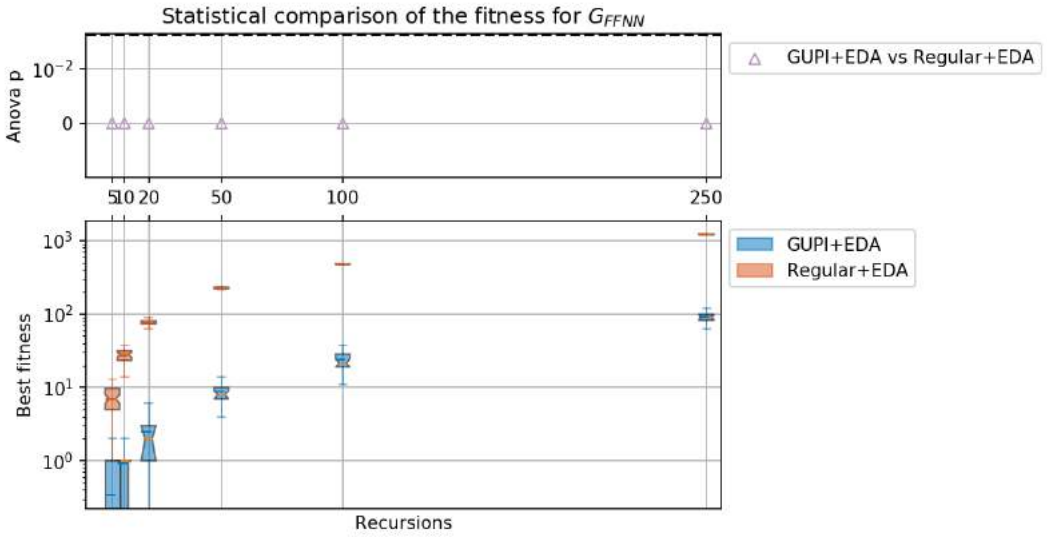

Figure 66: One-way ANOVA of the average fitness for G_{FFNN} to compare GUPI with a regular population initialization when using an estimation of distribution approach. 50 executions run per target derivation tree.

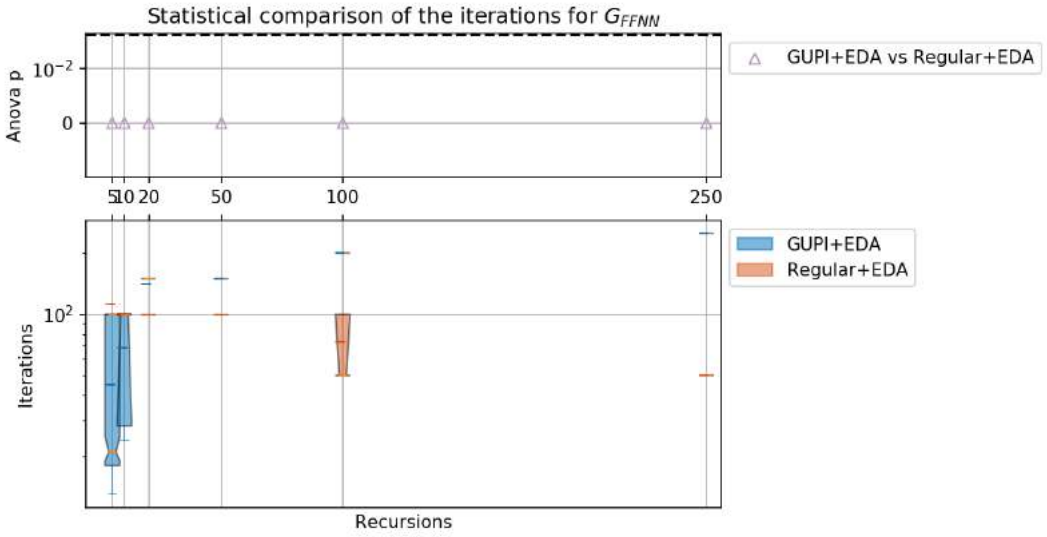

Figure 67: One-way ANOVA of the average iterations for G_{FFNN} to compare GUPI with a regular population initialization when using an estimation of distribution approach. 50 executions run per target derivation tree.

Deep Feed-forward Neural Network Optimization

Whigham Crossover

The performed experiments with G_{DFFNN} shows that there is not a fixed pattern in the best individual average fitness statistics when searching for derivation trees within **0** to **50** recursive derivations (Figure 74). GUPI outscores the regular

approach when searching for derivation trees of 5 recursive derivations (Figure 68), while the regular approach outscores GUPI for 10, 20 and 50 recursive derivations (Figure 69 to Figure 71). When the target derivation tree has more than 50 recursive derivations (Figure 72 and Figure 73), GUPI improves the average fitness of best individuals compare to the regular approach (Figure 74). Again, Figure 71 shows the first evidences of the difficulties of the regular population initialization approach due to the limitations to produce representative samples. Moreover, Figure 73 shows the premature convergence of the regular approach due to these same difficulties when looking for derivations trees with 250 recursive derivations. Statistical evidences are provided in Figure 73 to state there is a statistically significant difference between the approaches best fitness means ($p < 0.05$) for every target derivation. Figure 74 shows statistical differences for the number of generations, but these differences are mostly produced by the premature convergence of the approaches.

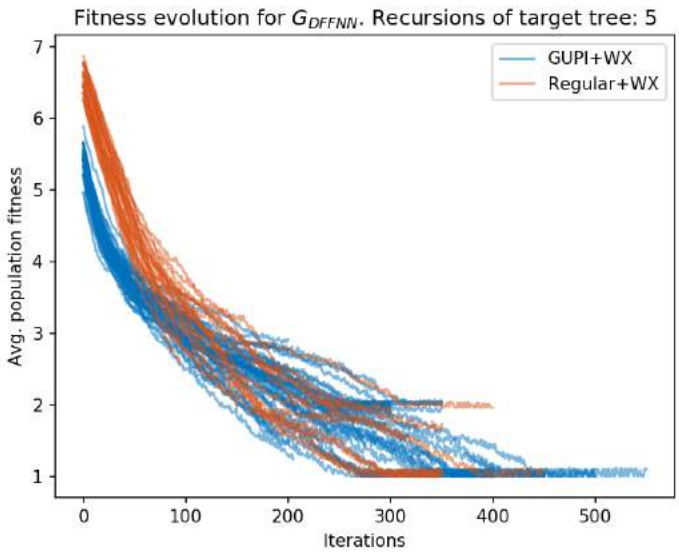

Figure 68: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{DFFNN} with 5 recursions is searched. 50 executions run.

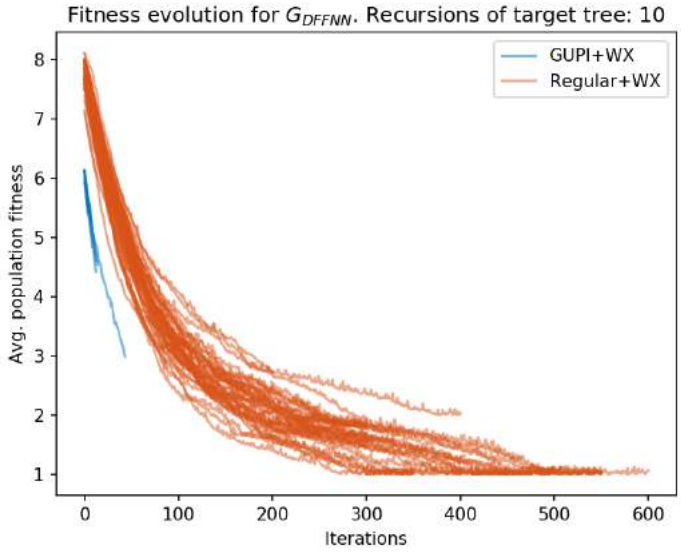

Figure 69: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{DFFNN} with 10 recursions is searched. 50 executions run.

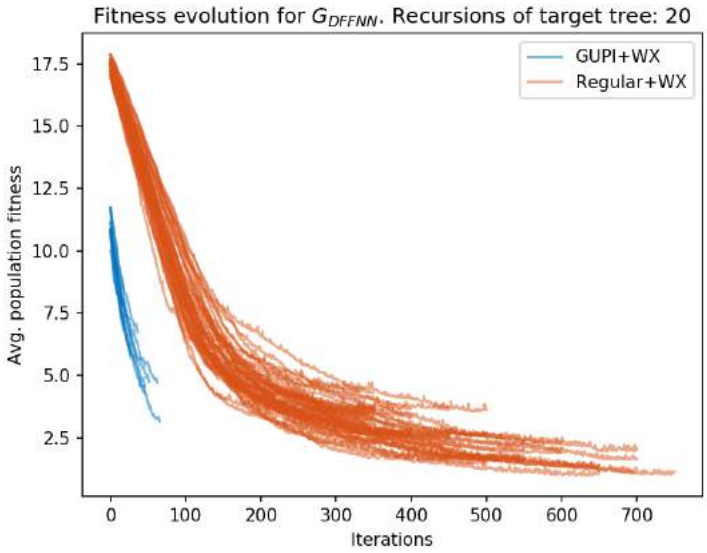

Figure 70: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{DFFNN} with 20 recursions is searched. 50 executions run.

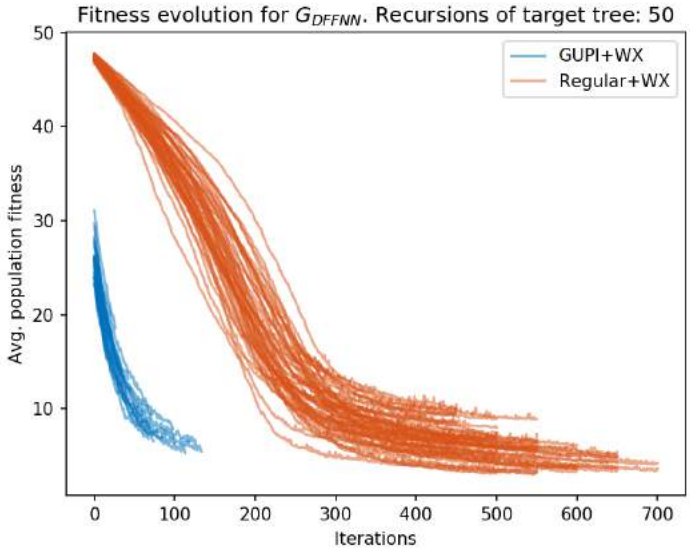

Figure 71: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{DFFNN} with 50 recursions is searched. 50 executions run.

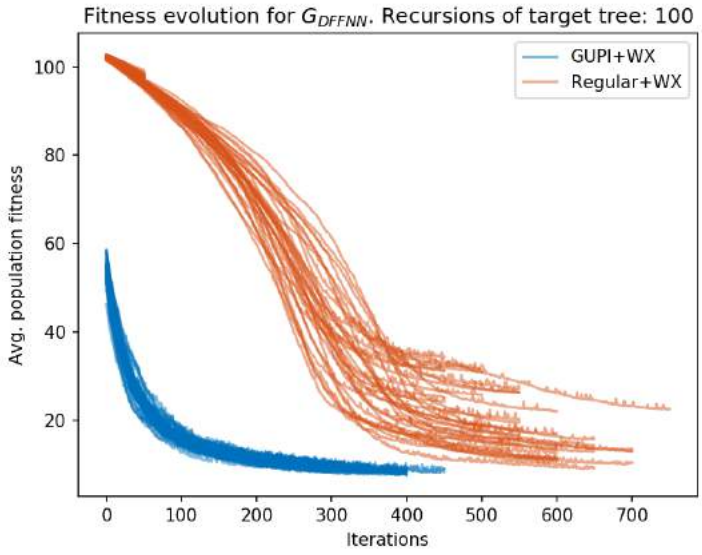

Figure 72: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{DFFNN} with 100 recursions is searched. 50 executions run.

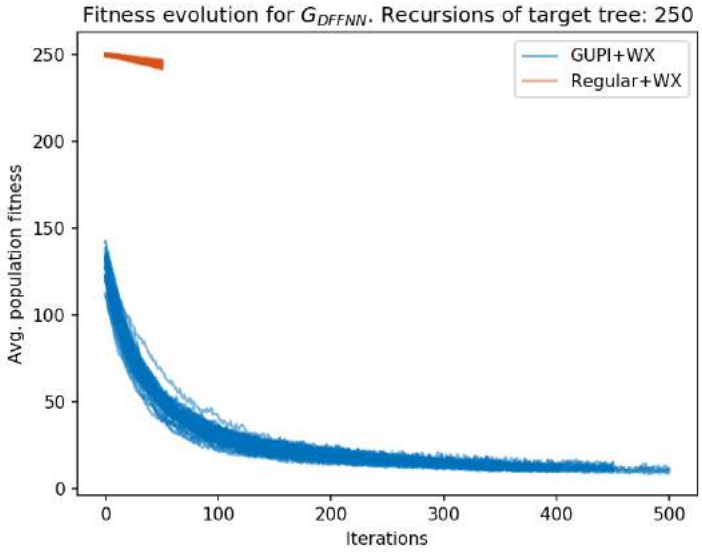

Figure 73: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{DFFNN} with 250 recursions is searched. 50 executions run.

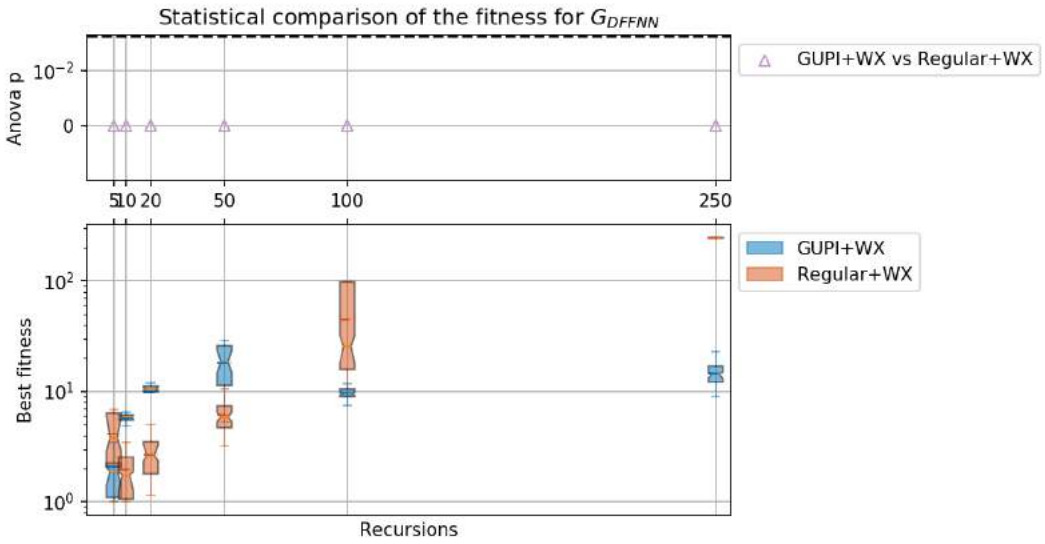

Figure 74: One-way ANOVA of the average fitness for G_{DFFNN} to compare GUPI with a regular population initialization when using a Whigham crossover. 50 executions run per target derivation tree.

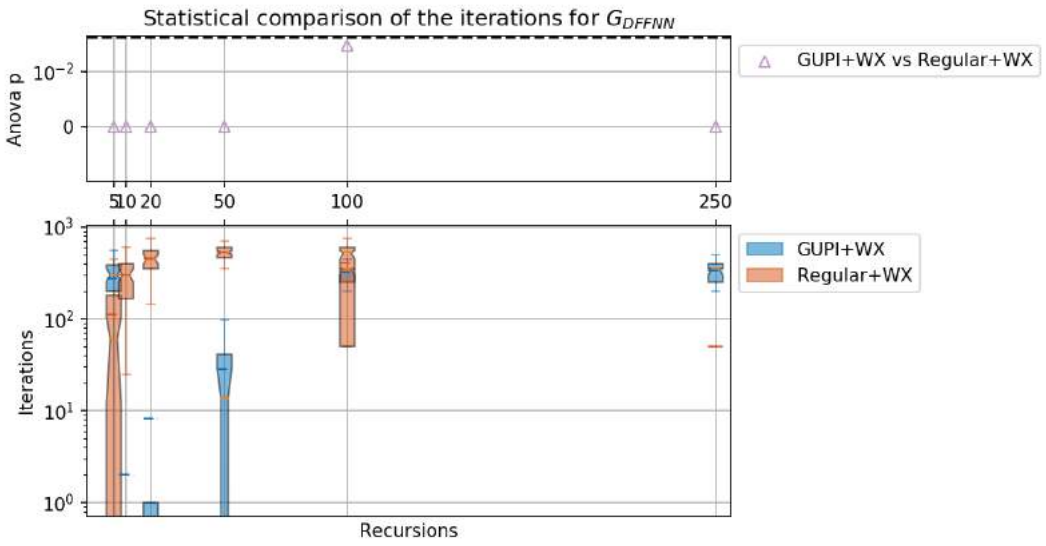

Figure 75: One-way ANOVA of the average iterations for G_{DFFNN} to compare GUPI with a regular population initialization when using a Whigham crossover. 50 executions run per target derivation tree.

Estimation of Distribution Approach

The performed experiments with the G_{DFFNN} in a GGGP-EDA approach shows GUPI significantly improves the evolutionary process independently of the target derivation tree (Figure 82). Figure 76 to Figure 80 shows the average fitness of the GUPI approach is significantly lower during the whole evolutionary process. Moreover, Figure 82 provides statistical evidences to state there is a statistically significant difference between the fitness means ($p < 0.05$). Moreover, Figure 83 shows GUPI approach is able to outperform or equal the regular approach in terms of generations.

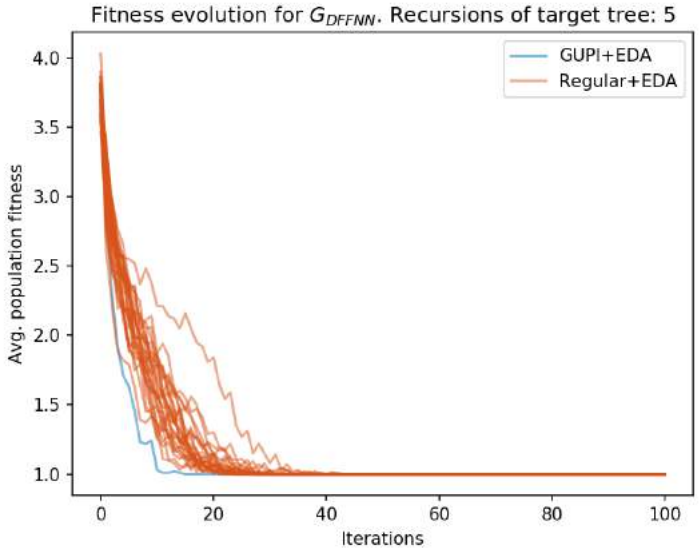

Figure 76: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{DFFNN} with 5 recursions is searched. 50 executions run.

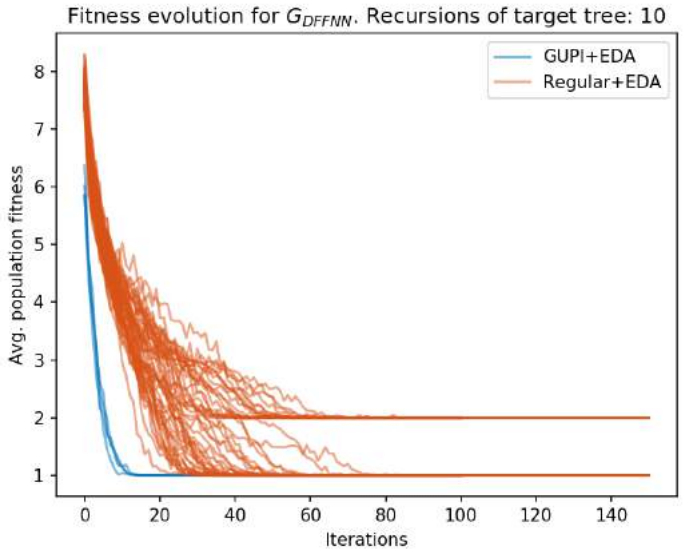

Figure 77: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{DFFNN} with 10 recursions is searched. 50 executions run.

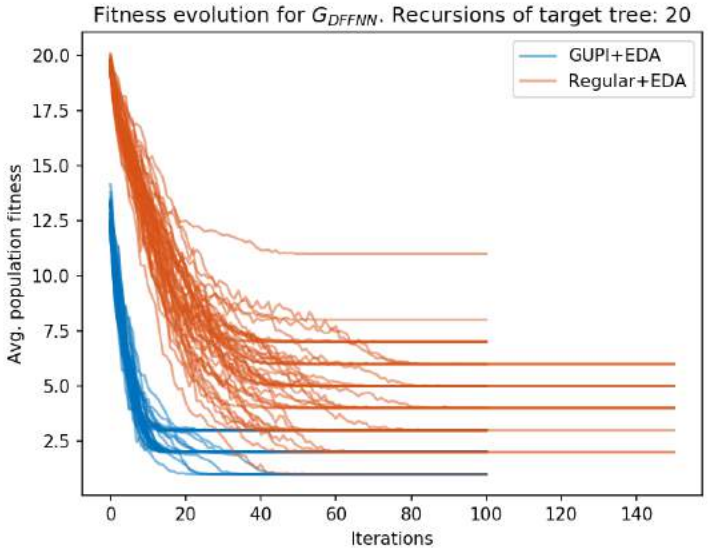

Figure 78: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{DFFNN} with 20 recursions is searched. 50 executions run.

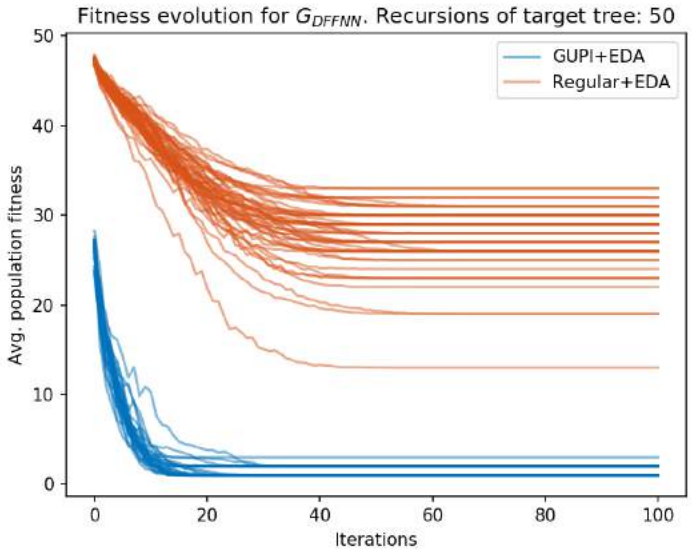

Figure 79: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{DFFNN} with 50 recursions is searched. 50 executions run.

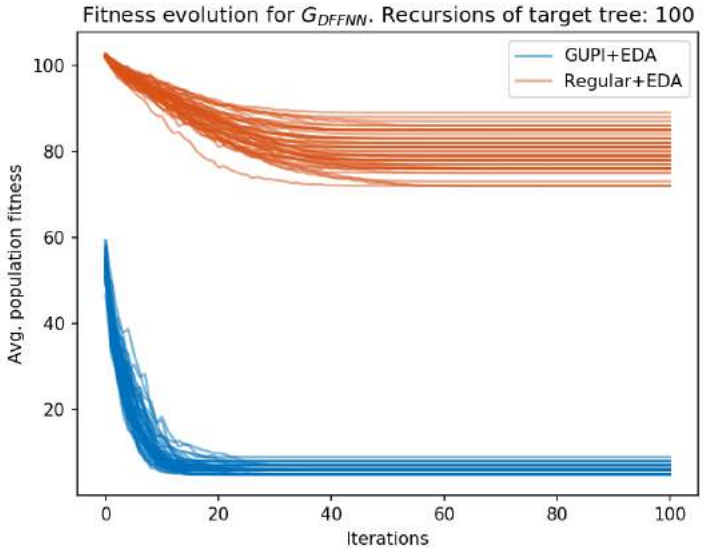

Figure 80: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{DFFNN} with 100 recursions is searched. 50 executions run.

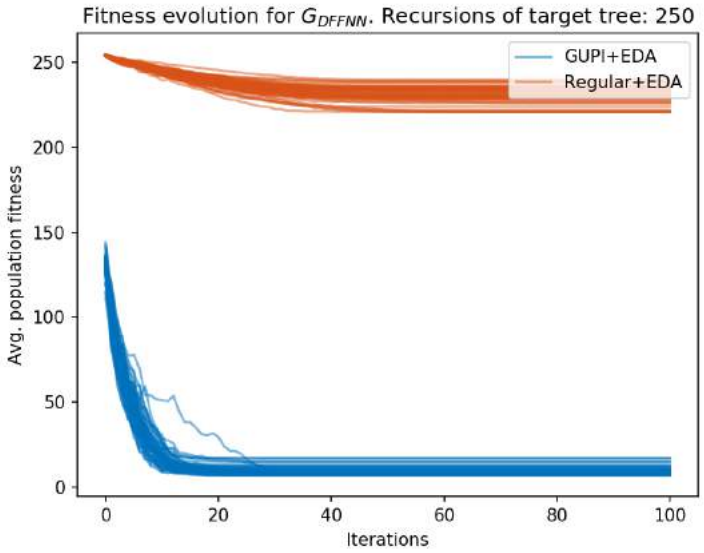

Figure 81: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{DFFNN} with 250 recursions is searched. 50 executions run.

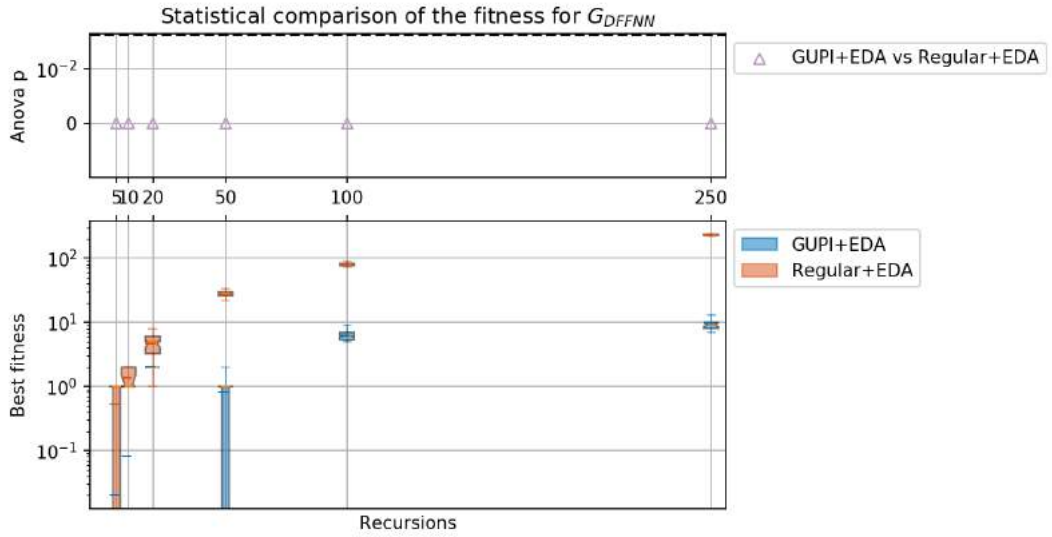

Figure 82: One-way ANOVA of the average fitness for G_{DFFNN} to compare GUPI with a regular population initialization when using an estimation of distribution approach. 50 executions run per target derivation tree.

Figure 83: One-way ANOVA of the average iterations for G_{DFFNN} to compare GUPI with a regular population initialization when using an estimation of distribution approach. 50 executions run per target derivation tree.

Rule-Based System Optimization

Whigham Crossover

The performed experiments with G_{RBS} shows that GUPI outperforms in terms of fitness the regular population initialization for all number of recursive derivations but 5 (Figure 90). Figure 87 to Figure 89 show how the regular approach

prematurely converges due to the population initialization difficulties to produce representative samples that lead the evolutionary when searching derivation trees with 50 or more recursive derivations. Evidences of these premature convergence are also provided in Figure 90 and Figure 91.

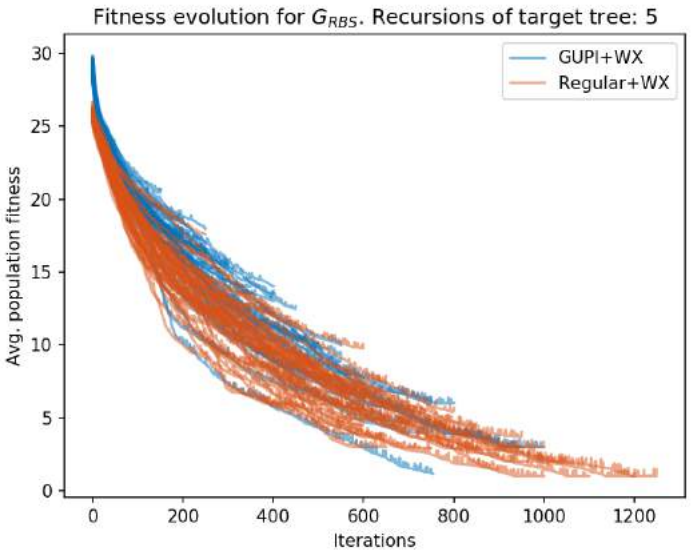

Figure 84: Fitness evolution comparison of GGGP CFG optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{RBS} with 5 recursions is searched. 50 executions run.

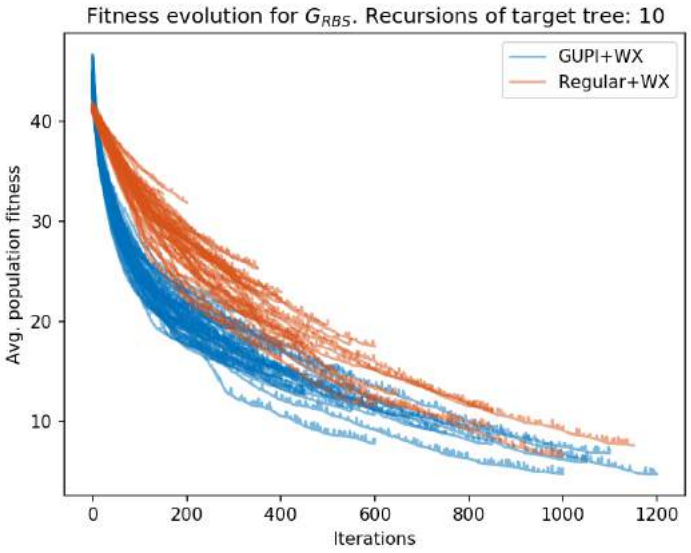

Figure 85: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{RBS} with 10 recursions is searched. 50 executions run.

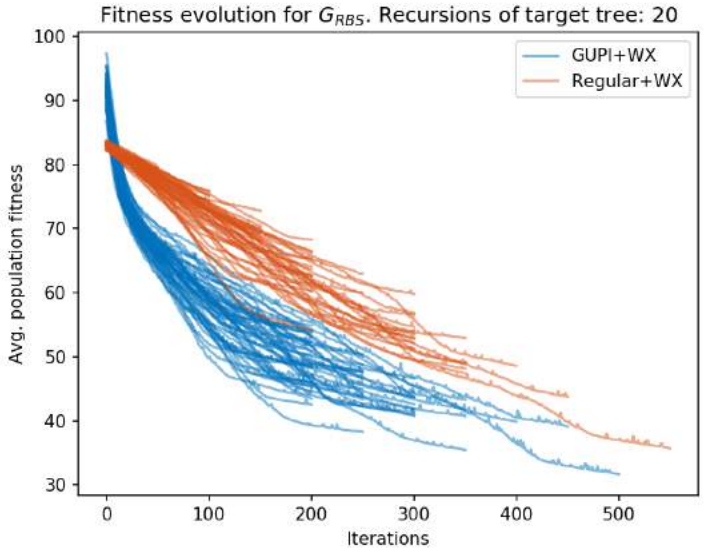

Figure 86: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{RBS} with 20 recursions is searched. 50 executions run.

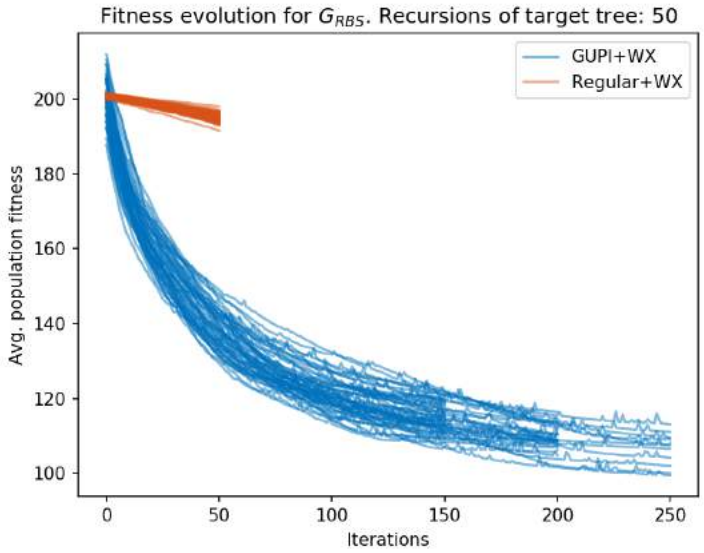

Figure 87: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{RBS} with 50 recursions is searched. 50 executions run.

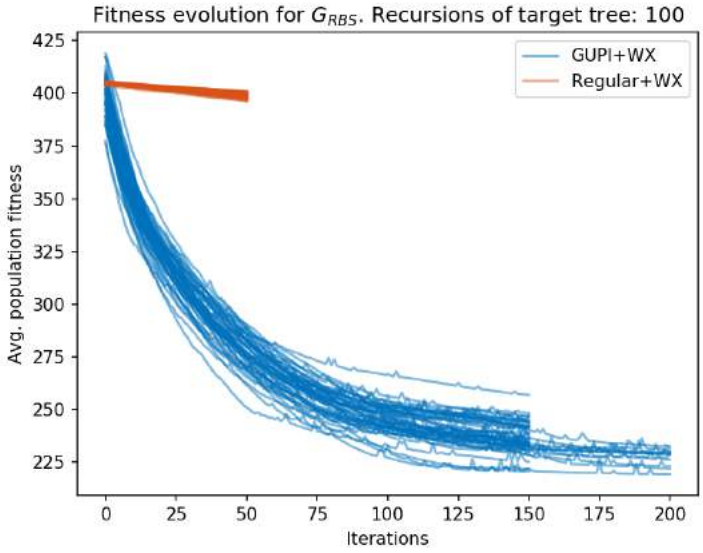

Figure 88: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{RBS} with 100 recursions is searched. 50 executions run.

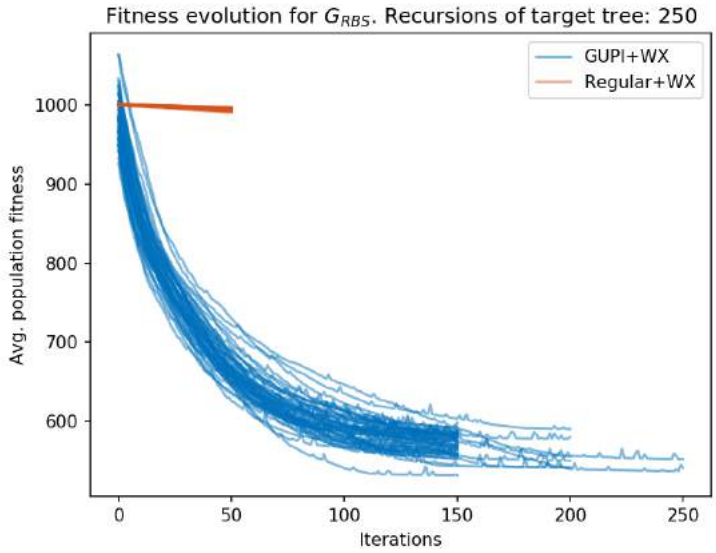

Figure 89: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{RBS} with 250 recursions is searched. 50 executions run.

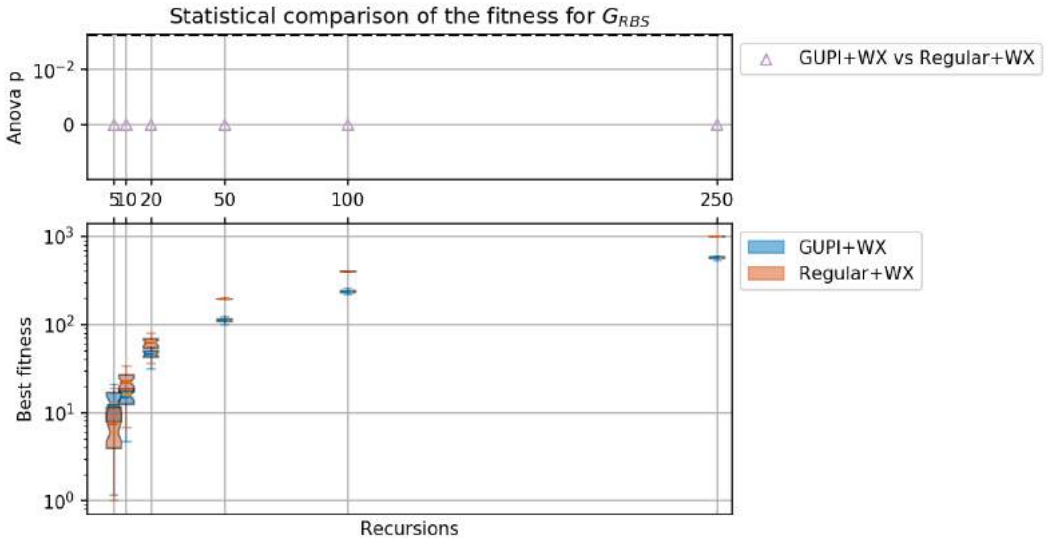

Figure 90: One-way ANOVA of the average fitness for G_{RBS} to compare GUPI with a regular population initialization when using a Whigham crossover. 50 executions run per target derivation tree.

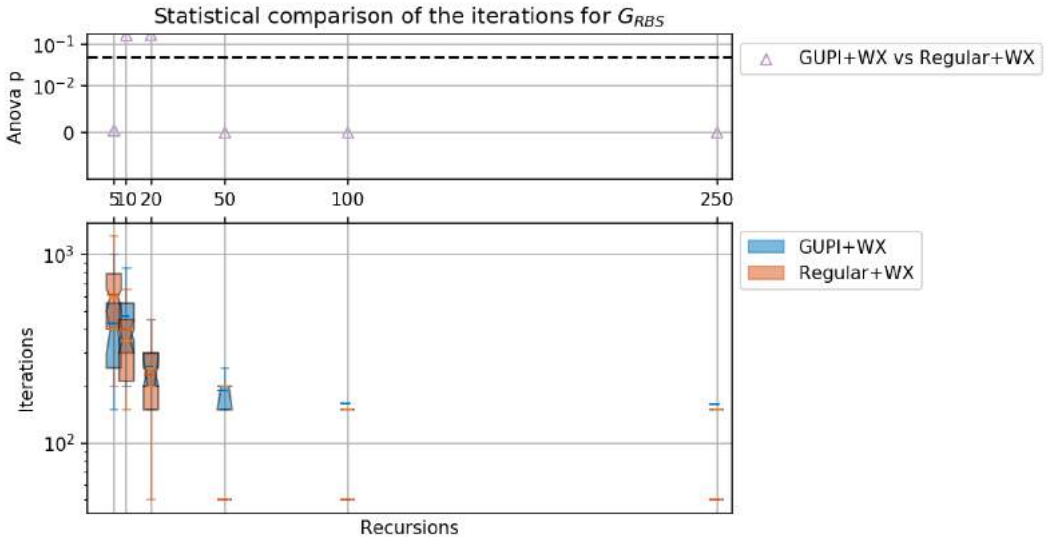

Figure 91: One-way ANOVA of the average iterations for G_{RBS} to compare GUPI with a regular population initialization when using a Whigham crossover. 50 executions run per target derivation tree.

Estimation of Distribution Approach

The performed experiments with the G_{RBS} in a GGGP-EDA approach shows GUPI significantly improves the evolutionary process when the target derivation tree has 10 or more recursive derivations (Figure 97). For 5 recursive derivations, the regular approach slightly improves the best average fitness due to the bias of the

regular population initialization towards derivation trees with few recursive derivation. Figure 92 show the evolutionary process is very similar for both approaches when looking for derivation trees with 5 recursive derivations. For 10 or more recursive derivations, Figure 93 to Figure 96 shows the average fitness of the GUPI approach is already significantly lower during the whole evolutionary process. Figure 97 provides statistical evidences to state there is a statistically significant difference between the fitness means ($p < 0.05$).

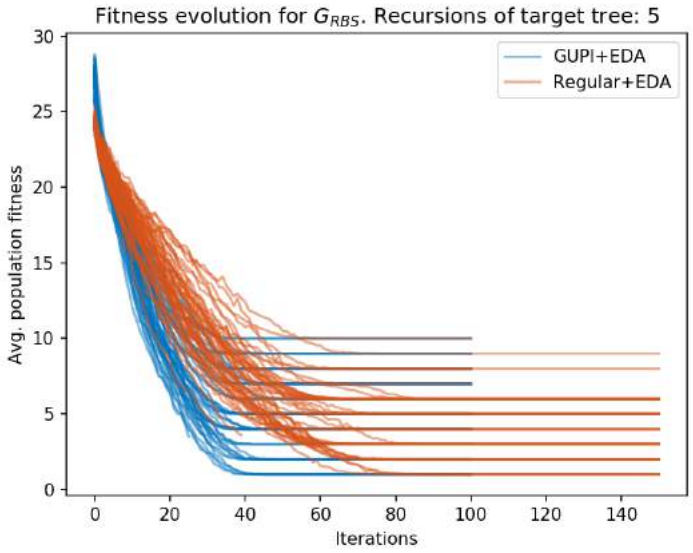

Figure 92: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{RBS} with 5 recursions is searched. 50 executions run.

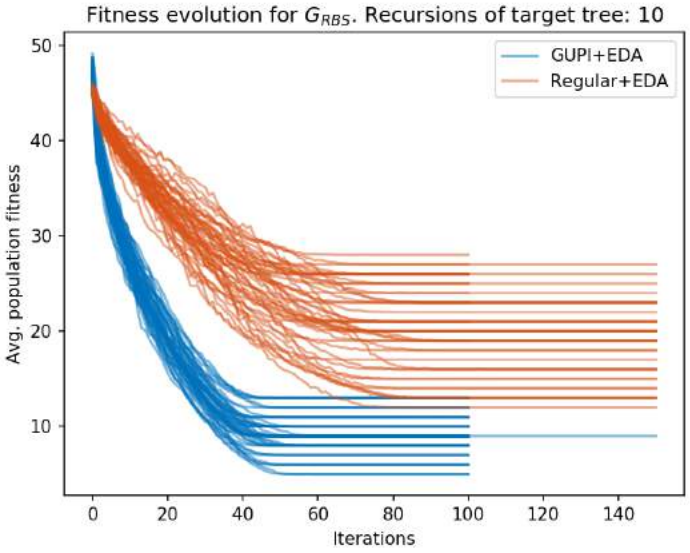

Figure 93: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{RBS} with 10 recursions is searched. 50 executions run.

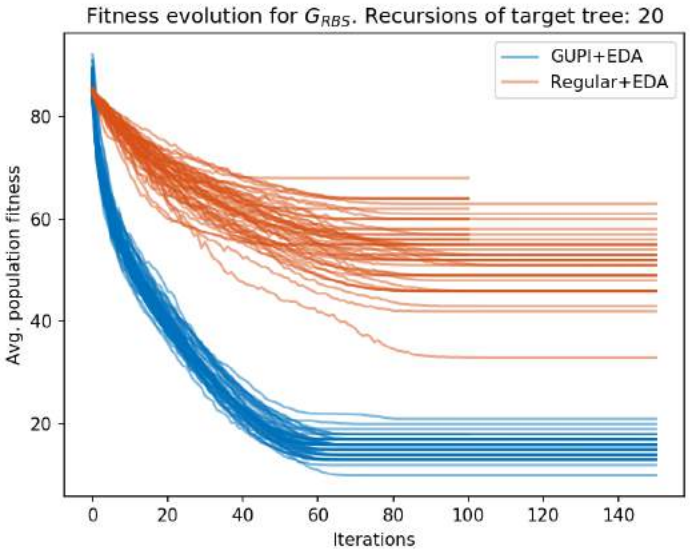

Figure 94: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{RBS} with 20 recursions is searched. 50 executions run.

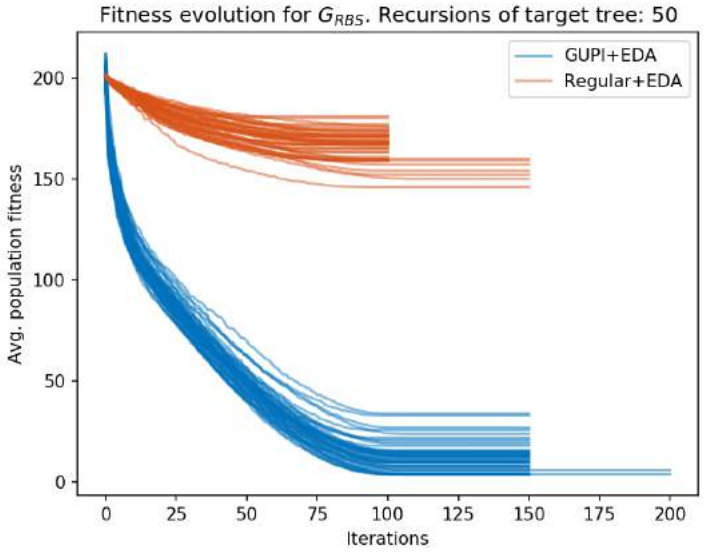

Figure 95: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{RBS} with 50 recursions is searched. 50 executions run.

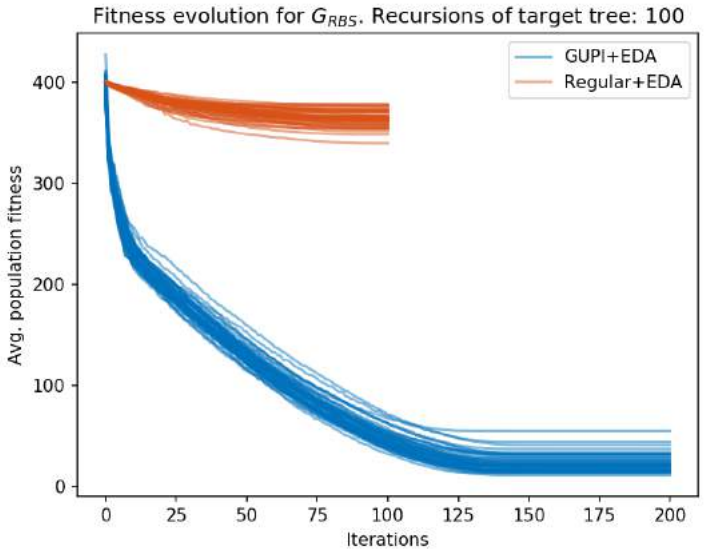

Figure 96: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{RBS} with 100 recursions is searched. 50 executions run.

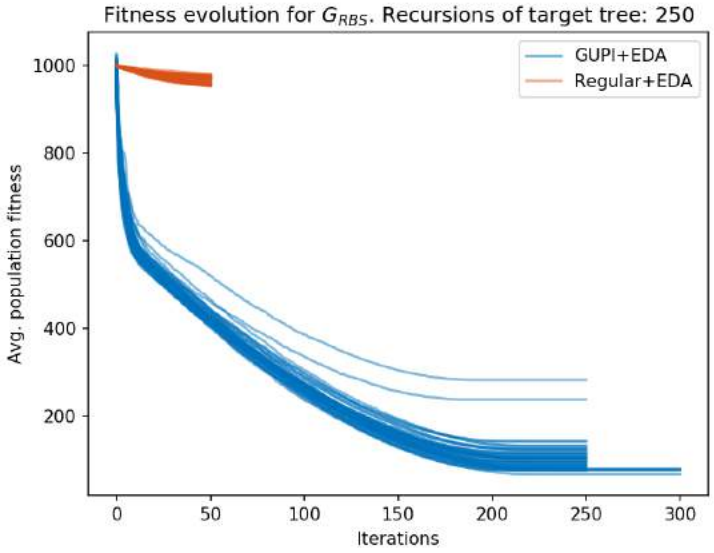

Figure 97: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{RBS} with 250 recursions is searched. 50 executions run.

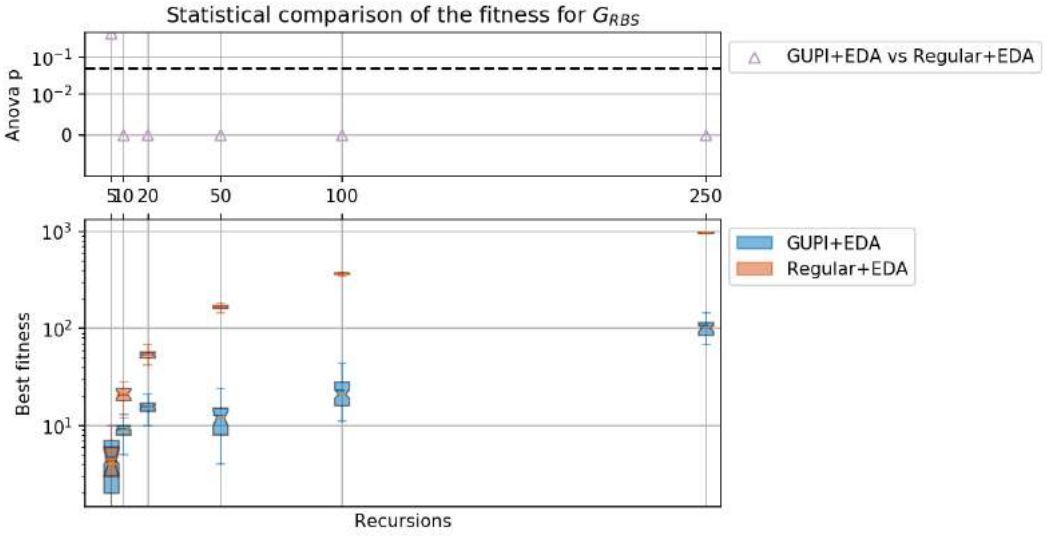

Figure 98: One-way ANOVA of the average fitness for G_{RBS} to compare GUPI with a regular population initialization when using an estimation of distribution approach. 50 executions run per target derivation tree.

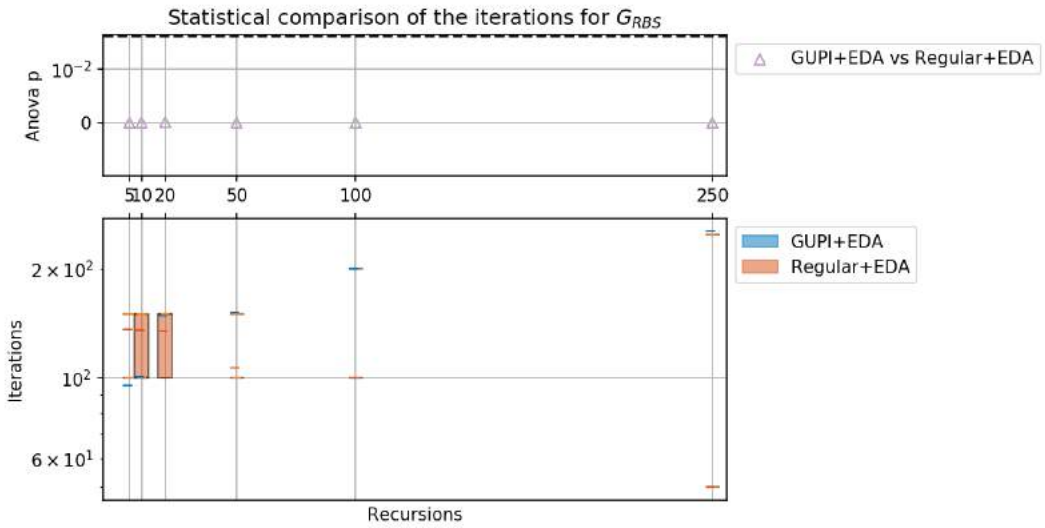

Figure 99: One-way ANOVA of the average iterations for G_{RBS} to compare GUPI with a regular population initialization when using an estimation of distribution approach. 50 executions run per target derivation tree.

Symbolic Regression Optimization

Whigham Crossover

Figure 100 to Figure 102 and Figure 106 shows a similar performance of the GGGP independently of the employed population initialization method when the target derivation tree of G_{SR} possess **20** or less recursive derivations. This happens due to the existence of a recursive production rule that generates twice the left-hand side nonterminal symbol. This production rule increases the probability of generating derivation trees with more recursive derivations when using the regular population initialization. Thus, it also produces representative samples when looking for derivation trees with higher number of recursive derivations. However, when the target derivation tree reaches **50** recursive derivations (Figure 103 to Figure 105), GUPI outscores the regular approach in terms of fitness and the mean is statistically proven to be lower (Figure 106). Evidences of the premature convergence of the regular approach when looking for derivations trees with **100** and **250** are shown Figure 104, Figure 105 and Figure 107.

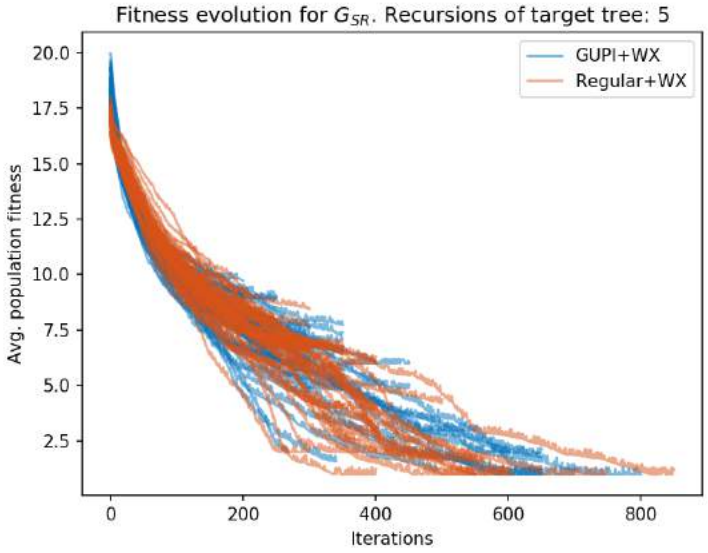

Figure 100: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{SR} with 5 recursions is searched. 50 executions run.

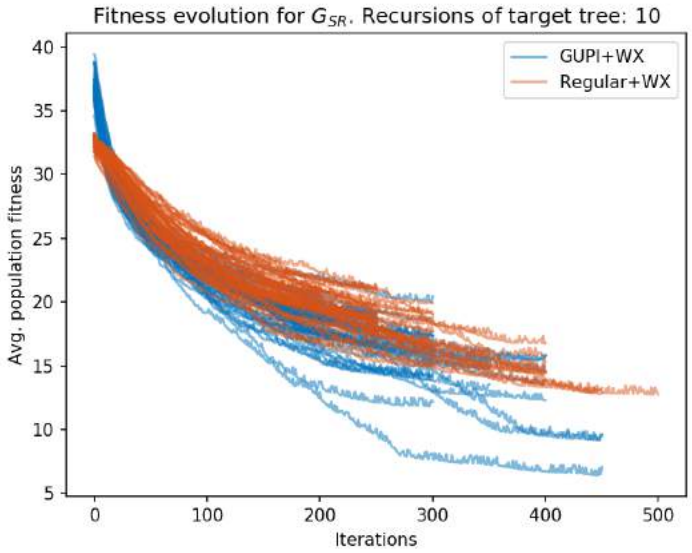

Figure 101: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{SR} with 10 recursions is searched. 50 executions run.

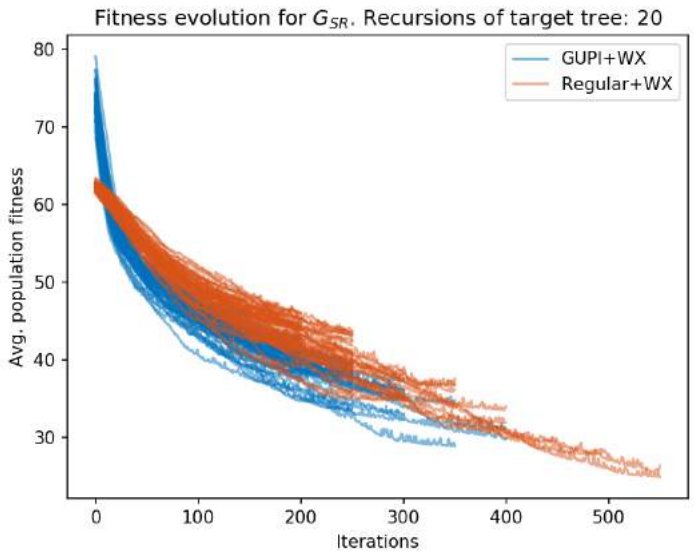

Figure 102: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{SR} with 20 recursions is searched. 50 executions run.

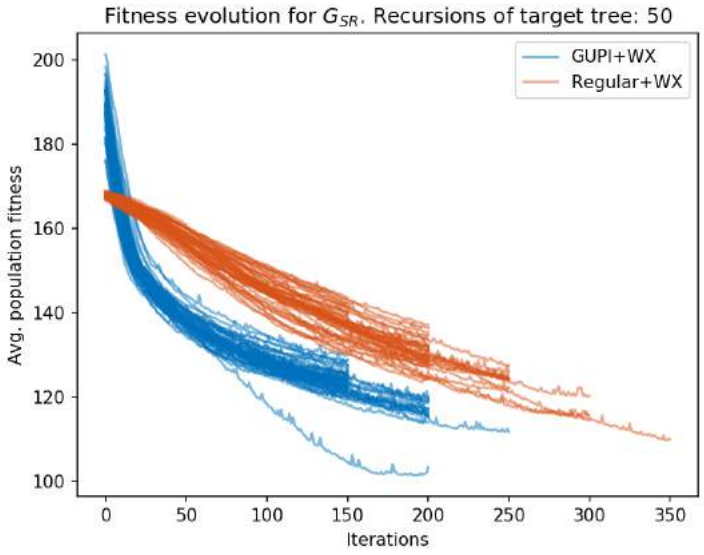

Figure 103: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{SR} with 50 recursions is searched. 50 executions run.

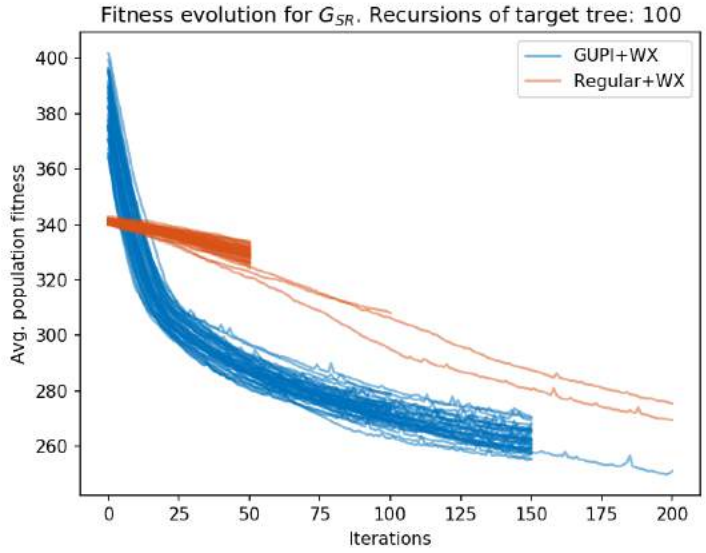

Figure 104: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{SR} with 100 recursions is searched. 50 executions run.

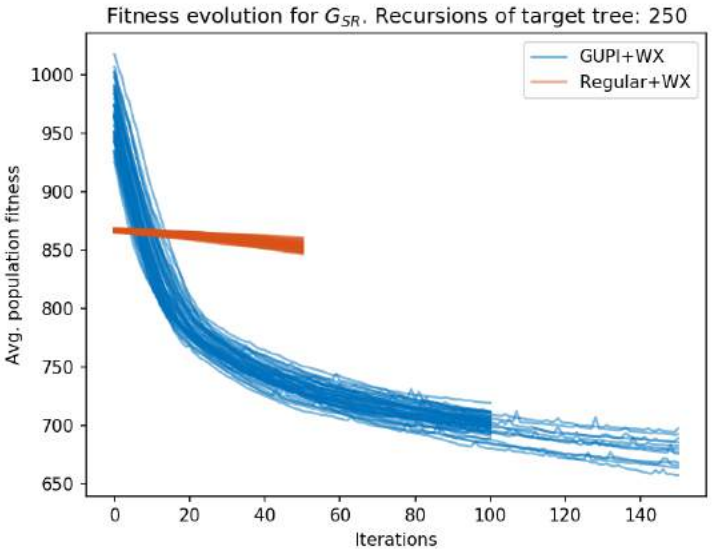

Figure 105: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{SR} with 250 recursions is searched. 50 executions run.

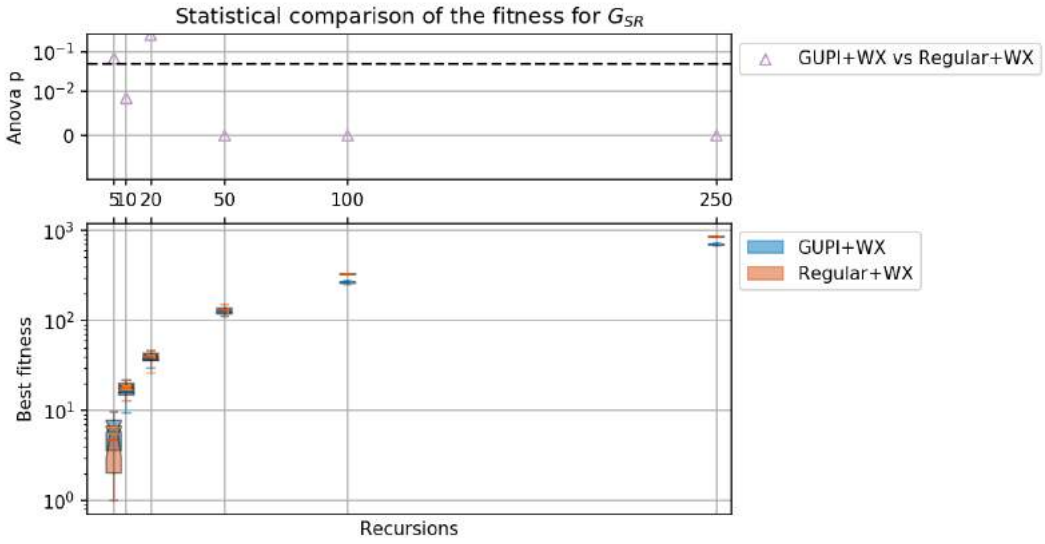

Figure 106: One-way ANOVA of the average fitness for G_{SR} to compare GUPI with a regular population initialization when using a Whigham crossover. 50 executions run per target derivation tree.

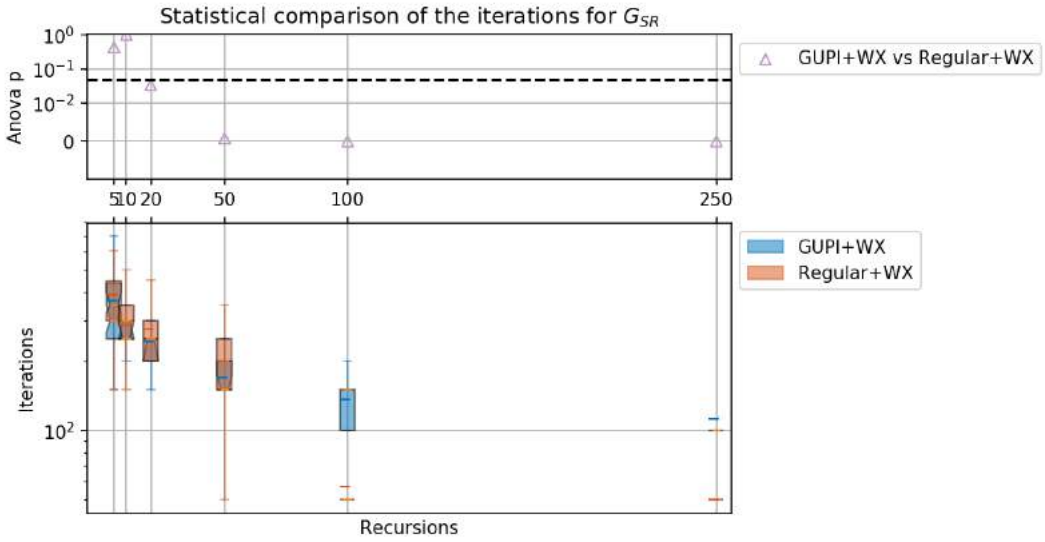

Figure 107: One-way ANOVA of the average iterations for G_{SR} to compare GUPI with a regular population initialization when using a Whigham crossover. 50 executions run per target derivation tree.

Estimation of Distribution Approach

The performed experiments with the SR CFG in a GGGP-EDA approach shows GUPI significantly improves the evolutionary process for all number of recursive derivation (Figure 114) but 5, which does not show any difference (Figure 108) Figure 109 to Figure 113 shows the average fitness of the GUPI approach is already

lower during the whole evolutionary process. Moreover, Figure 114 provides statistical evidences ($p < 0.05$) to state there is a statistically significant difference between the fitness means.

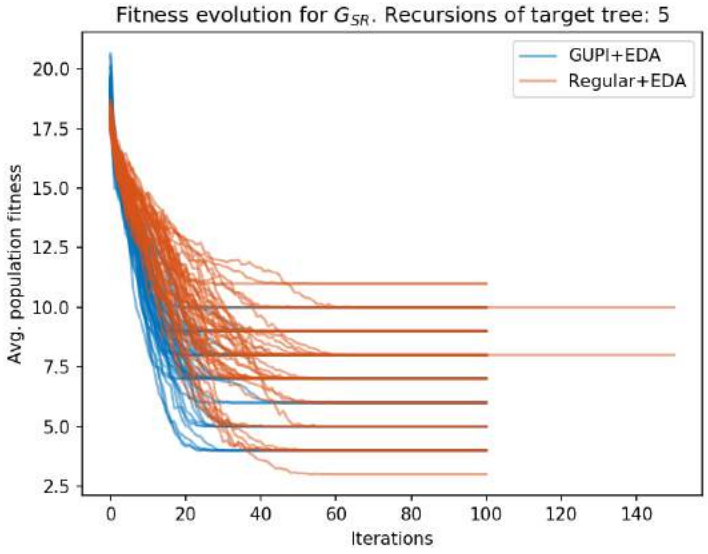

Figure 108: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{SR} with 5 recursions is searched. 50 executions run.

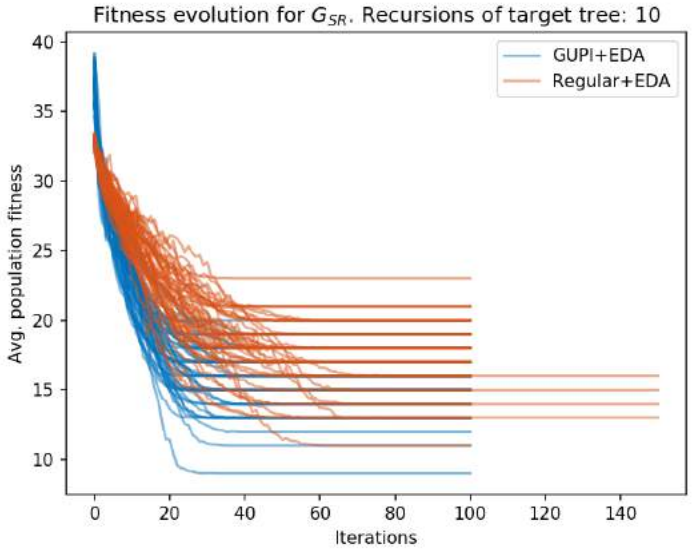

Figure 109: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{SR} with 10 recursions is searched. 50 executions run.

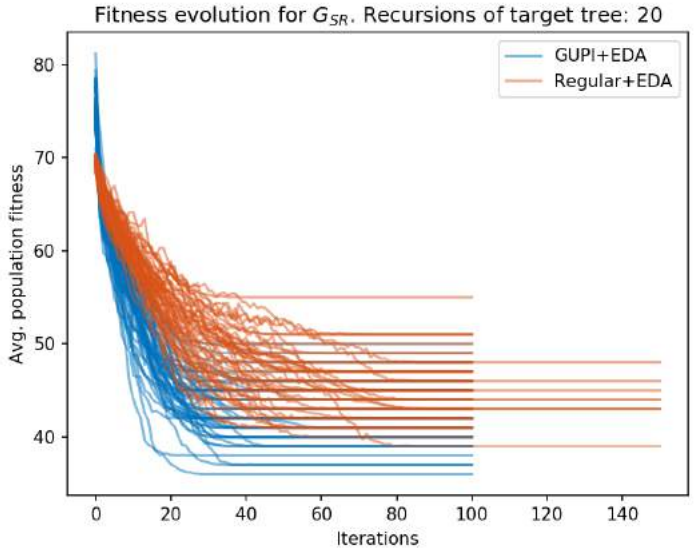

Figure 110: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{SR} with 20 recursions is searched. 50 executions run.

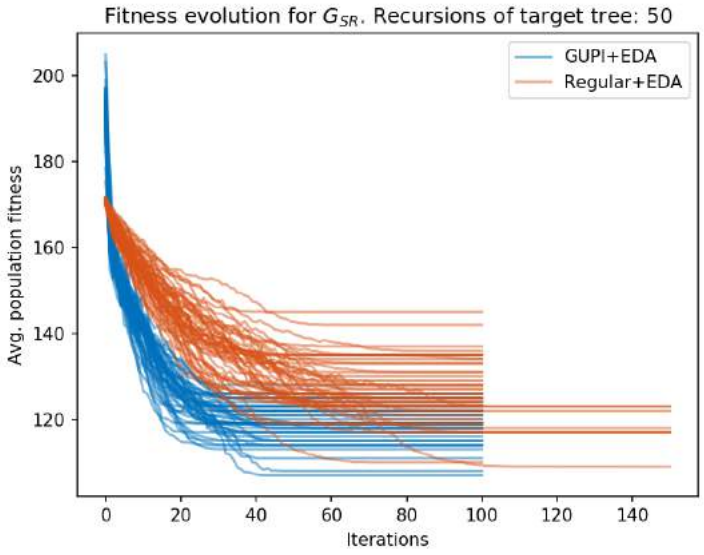

Figure 111: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{SR} with 50 recursions is searched. 50 executions run.

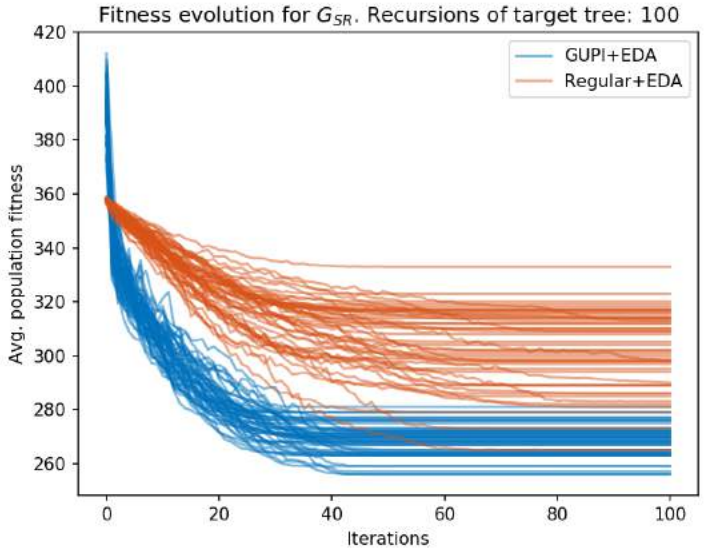

Figure 112: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{SR} with 100 recursions is searched. 50 executions run.

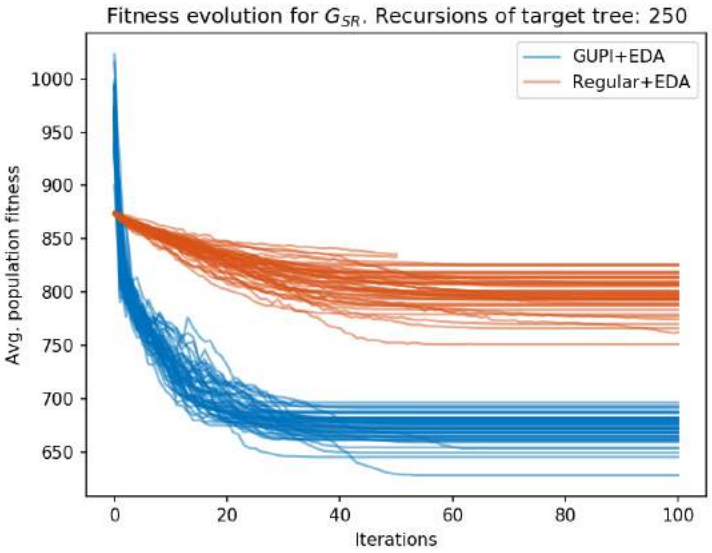

Figure 113: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{SR} with 250 recursions is searched. 50 executions run.

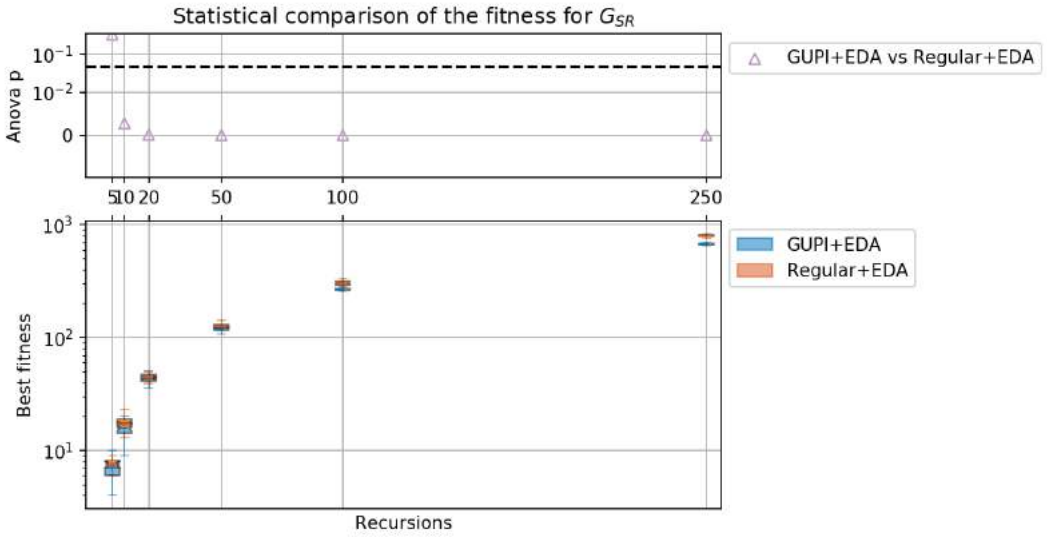

Figure 114: One-way ANOVA of the average fitness for G_{SR} to compare GUPI with a regular population initialization when using an estimation of distribution approach. 50 executions run per target derivation tree.

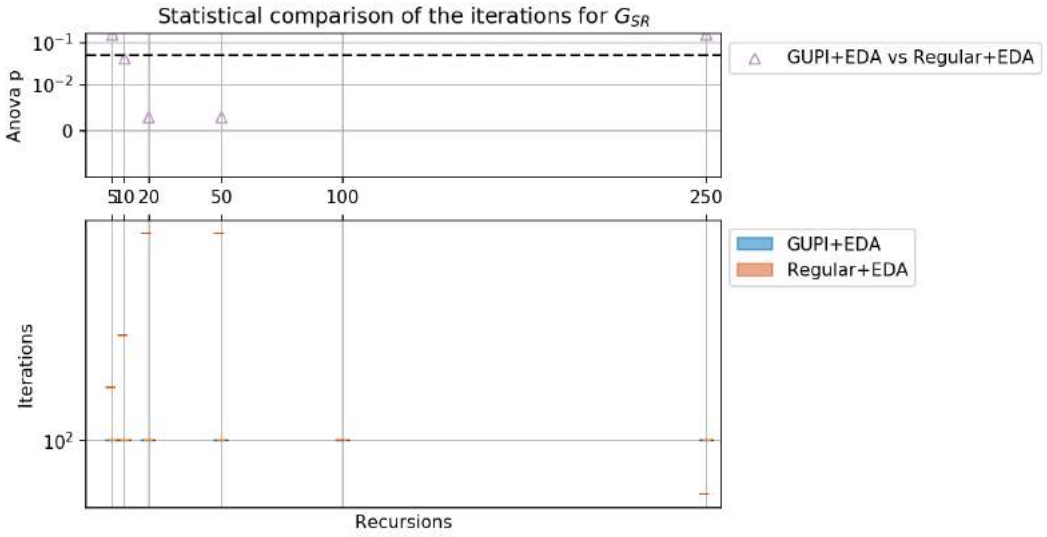

Figure 115: One-way ANOVA of the average iterations for G_{SR} to compare GUPI with a regular population initialization when using an estimation of distribution approach. 50 executions run per target derivation tree.

Boolean Optimization Problem

Whigham Crossover

Similar results to the G_{SR} experiments are obtained when looking for derivation trees of G_{BP} since it also contains a production rule that generates twice the left-hand side nonterminal symbol. Again, both approaches show similar performance

independently of the employed population initialization method when the target derivation tree has 10 recursive derivations or less (Figure 116 and Figure 117). However, the regular approach starts showing some difficulties to guide the evolutionary process for 20 recursive derivations (Figure 118). Then, GUPI shows better average fitness for 20 or more recursive derivations (Figure 118 to Figure 121) in the Figure 122. Moreover, GUPI reduces the necessary generations to obtain an optimal solution compare to the regular approach when the target derivation tree possesses 50 or more recursive derivations (Figure 123).

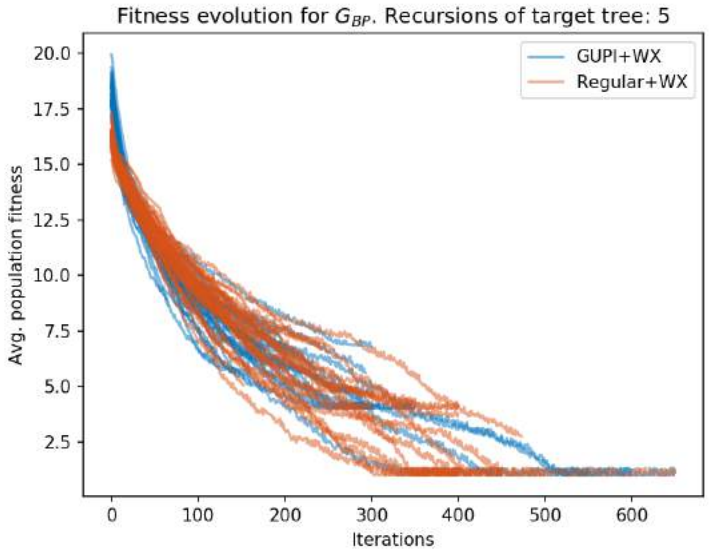

Figure 116: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{BP} with 5 recursions is searched. 50 executions run.

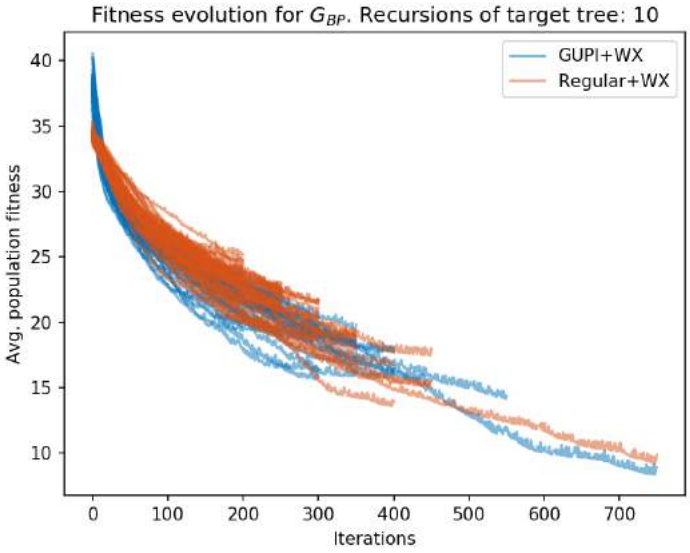

Figure 117: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{BP} with 10 recursions is searched. 50 executions run.

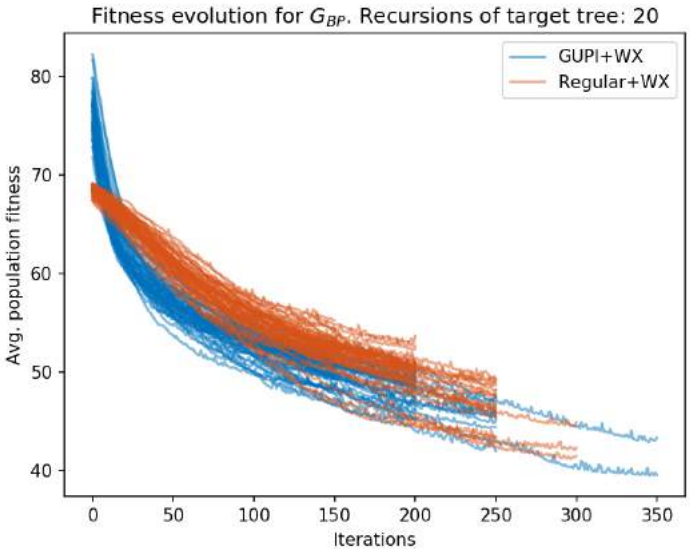

Figure 118: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{BP} with 20 recursions is searched. 50 executions run.

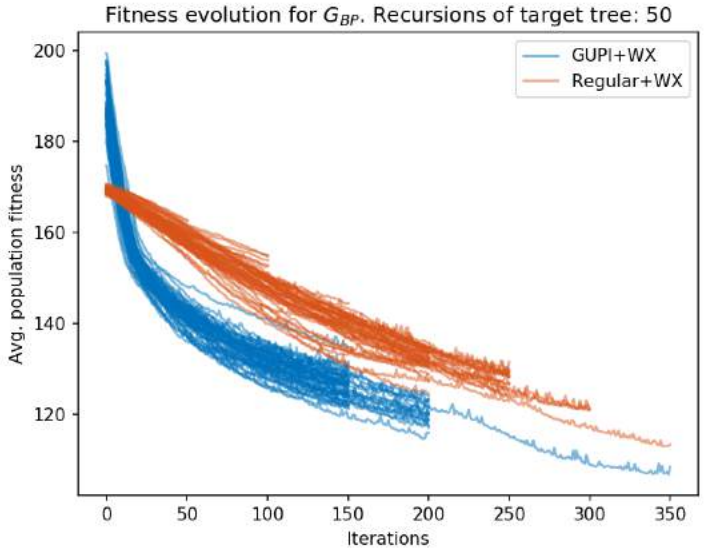

Figure 119: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{BP} with 50 recursions is searched. 50 executions run.

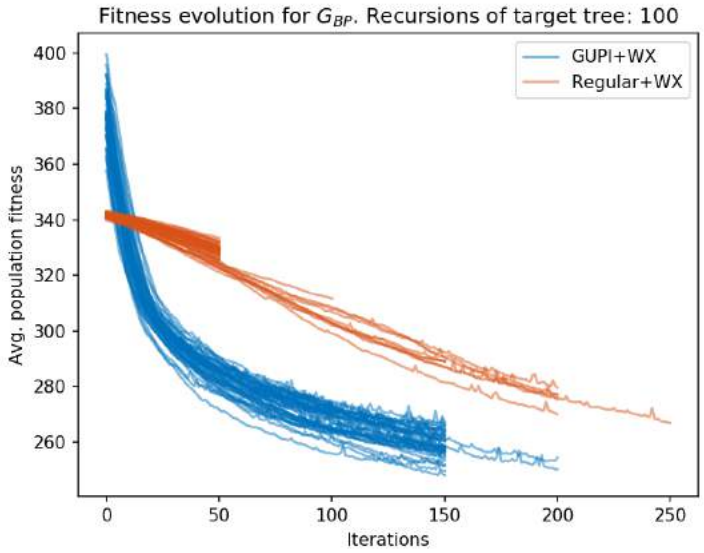

Figure 120: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{BP} with 100 recursions is searched. 50 executions run.

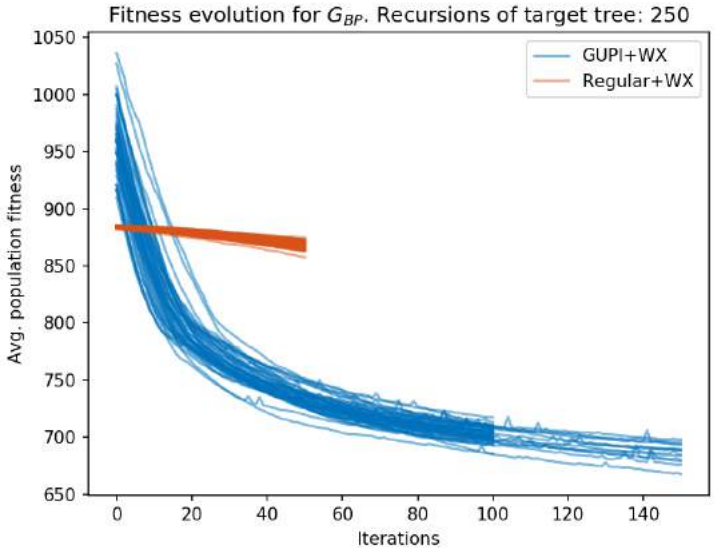

Figure 121: Fitness evolution comparison of GGGP optimization when using GUPI or a regular population initialization with a WX crossover. A derivation tree of G_{BP} with 250 recursions is searched. 50 executions run.

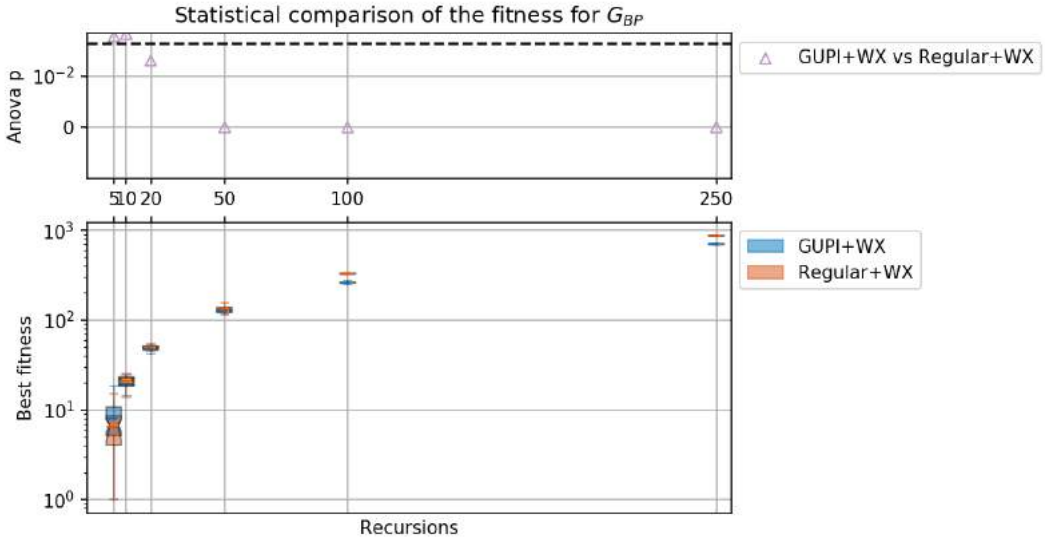

Figure 122: One-way ANOVA of the average fitness for G_{BP} to compare GUPI with a regular population initialization when using a Whigham crossover. 50 executions run per target derivation tree.

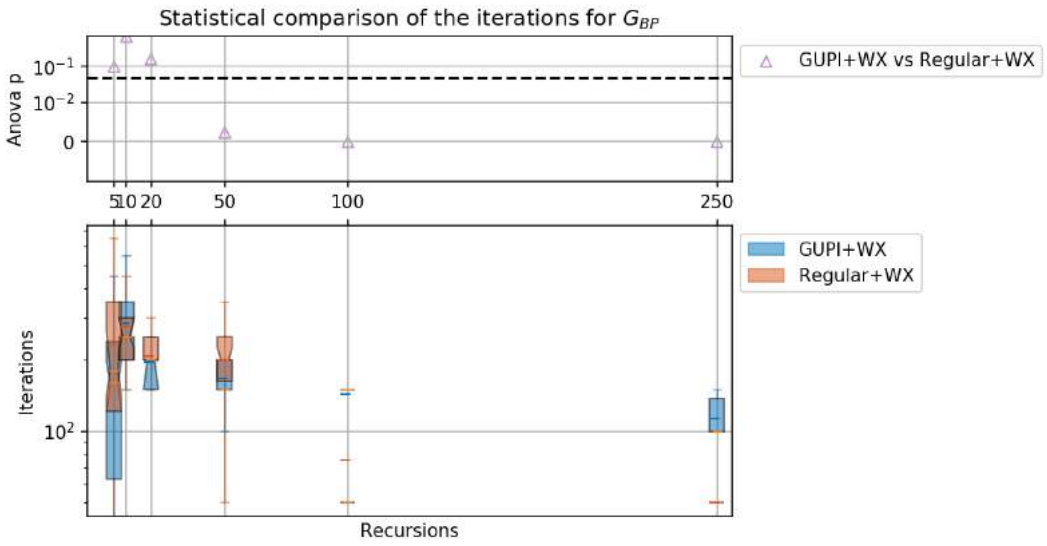

Figure 123: One-way ANOVA of the average iterations for G_{BP} to compare GUPI with a regular population initialization when using a Whigham crossover. 50 executions run per target derivation tree.

Estimation of Distribution Approach

The performed experiments with the G_{BP} in a GGGP-EDA approach shows GUPI significantly improves the average best fitness when the target derivation tree has more than 10 recursive derivations (Figure 130). For these cases, Figure 126 to Figure 128 shows the average fitness of the GUPI approach is already lower during the whole evolutionary process. Moreover, Figure 130 provides statistical evidences ($p < 0.05$) to state there is a statistically significant difference between the fitness means. When the target derivation tree has 10 or less recursive derivations, the whole evolutionary process is very similar (Figure 124 and Figure 125) and no evidence is provided to state there is a statistically significant difference between the fitness means (Figure 130). However, in this case the GUPI approach outperforms in terms of generations the regular approach in two of the three cases (Figure 131).

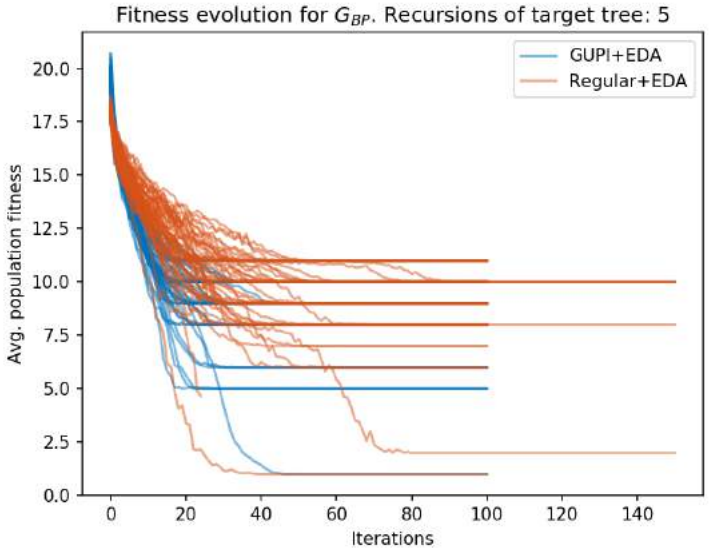

Figure 124: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{BP} with 5 recursions is searched. 50 executions run.

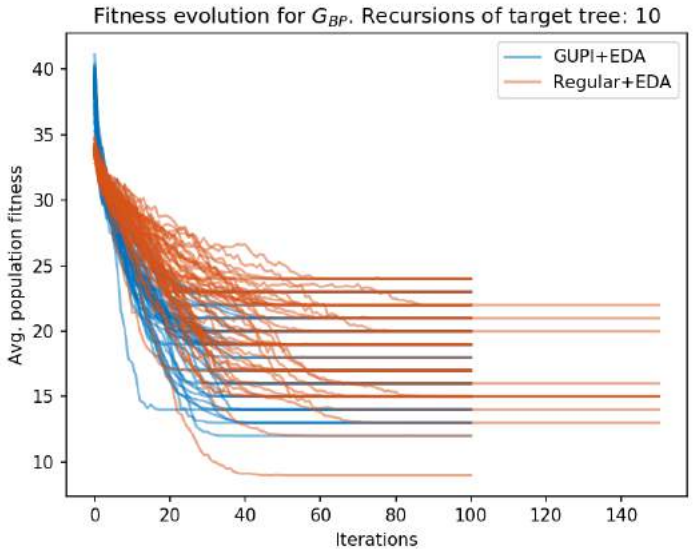

Figure 125: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{BP} with 10 recursions is searched. 50 executions run.

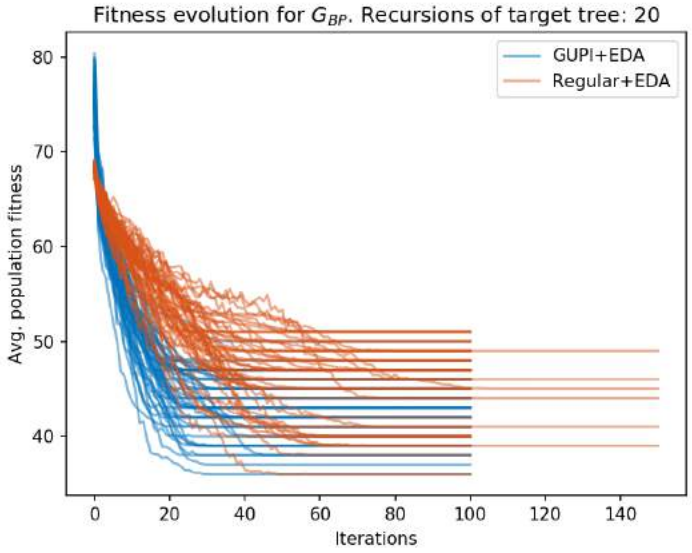

Figure 126: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{BP} with 20 recursions is searched. 50 executions run.

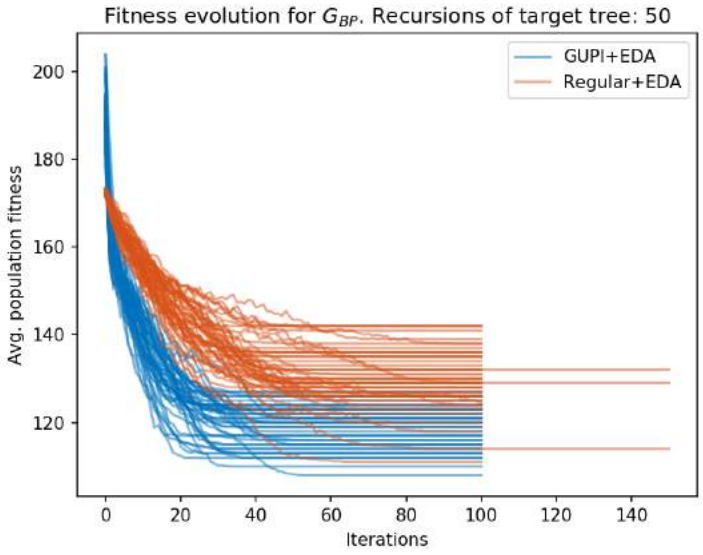

Figure 127: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{BP} with 50 recursions is searched. 50 executions run.

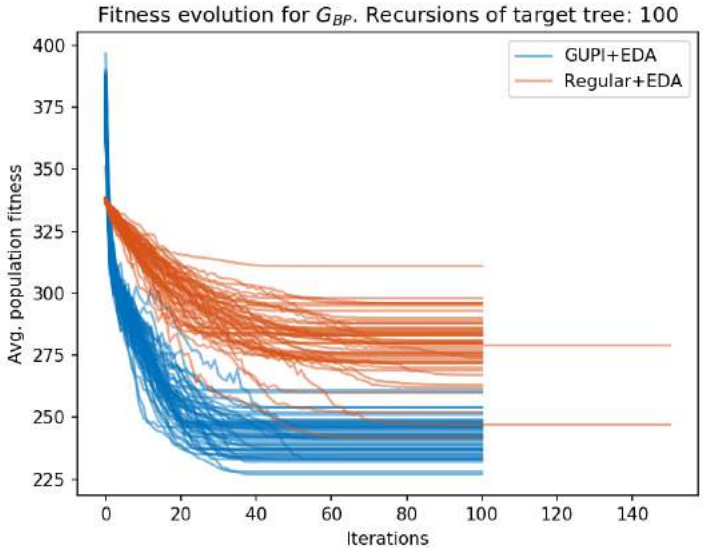

Figure 128: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{BP} with 100 recursions is searched. 50 executions run.

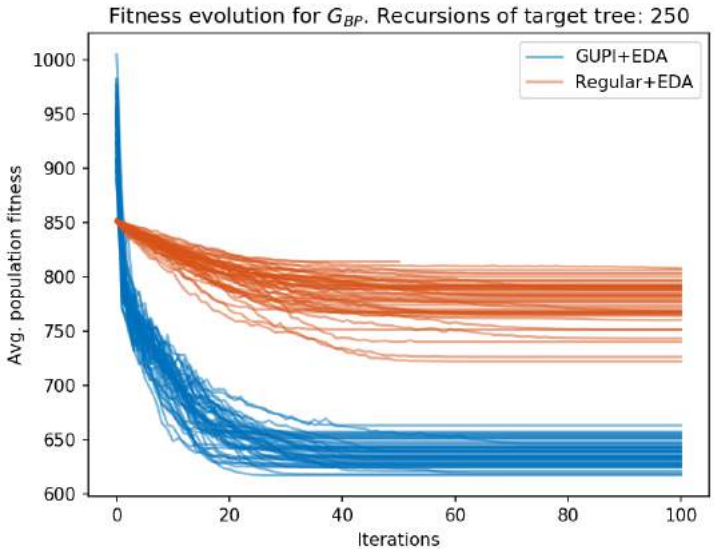

Figure 129: Fitness evolution comparison of GGGP-EDA optimization when using GUPI or a regular population initialization. A derivation tree of G_{BP} with 250 recursions is searched. 50 executions run.

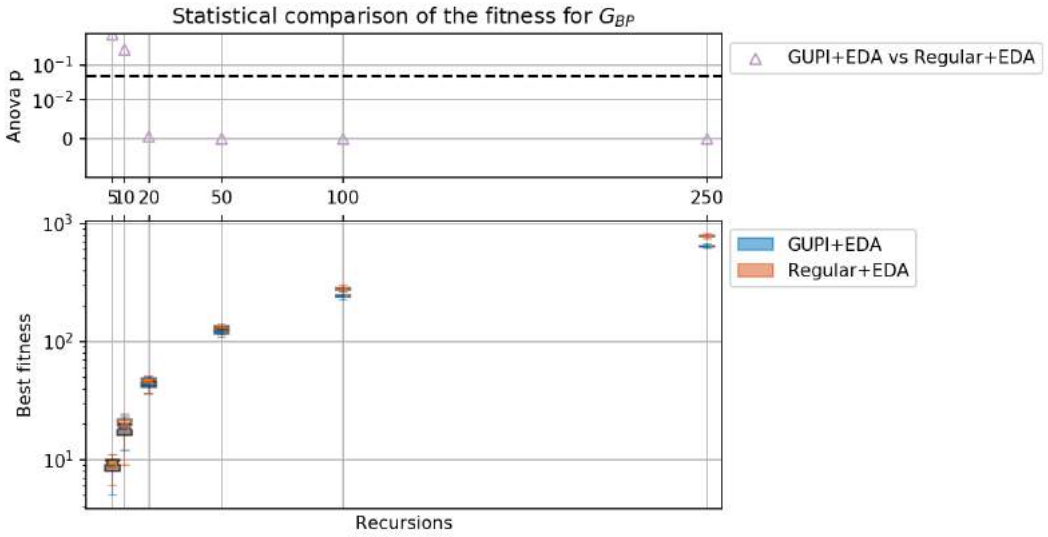

Figure 130: One-way ANOVA of the average fitness for G_{BP} to compare GUPI with a regular population initialization when using an estimation of distribution approach. 50 executions run per target derivation tree.

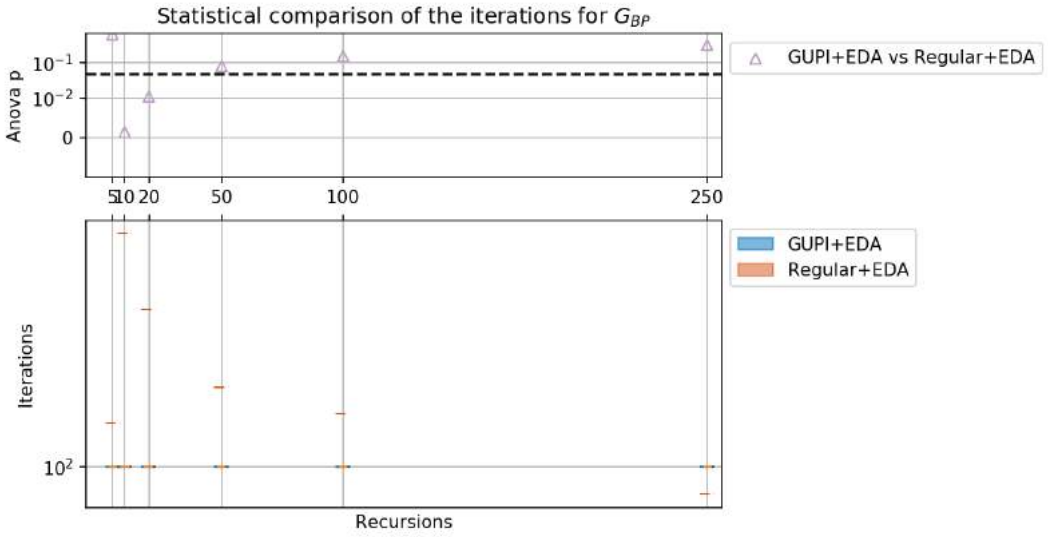

Figure 131: One-way ANOVA of the average iterations for G_{BP} to compare GUPI with a regular population initialization when using an estimation of distribution approach. 50 executions run per target derivation tree.

6. Estimation of Distribution Crossover

A new crossover operator based on the estimation of distributions is presented. This operator is called estimation of distribution crossover (EDX). The EDX is a crossover operator that borrows ideas from EDAs to generate a new offspring. As

EDAs does, the EDX estimates the population distribution to produce a probabilistic model that encodes the characteristics of most promising individuals to later generate a new offspring. However, the EDX acts as crossover operator and instead of producing a whole new generation that replaces the previous one, it produces a small set of individuals that replaces some of the worst-fit population individuals. This feature improves the population diversity preservation so that the likelihood of premature convergence is reduced. A smoothing method is also provided to enhance the local exploration capabilities.

The dynamic stochastic CFG presented for GUPI (12) is extended to represent a probabilistic model that approximates the EDX population distribution. The CFG search space is represented as a tree graph where each node n represents a derivation tree node, and the children branches, each possible derivation from a nonterminal node $n_A, A \in V$. This graph-like representation of the CFG search space that encodes the probabilistic model is called CFG expansion (CFGE). In addition to the probabilities $P(r)$ (12) associated to choose a production rule r within a nonterminal production set R_A, a new probability is assigned to each branch of the CFGE graph. This new probability $\rho(r_{n_A})$, where r_{n_A} is the rule r such that $r \in R_A$ for that node n_A, encodes the relative frequency of a specific derivation produced by r within all possible derivations starting from a specific derivation tree node n_A.

The CFGE probabilistic model construction 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 starts after the selection operator chooses the subset of individuals that will be used to build the probabilistic model.

1) The calculation of the frequencies traverses every selected derivation from the root to the leaves. During this process, it checks what production rule has been applied in every node n and store this information in the CFGE. Every time a production rule r is selected in a nonterminal node n_A, the associated weight $\omega_{r_{n_A}}$ in the CFGE (which is initially setup to 0) is

increased by 1. Thus, once every selected derivation tree has been traversed, the CFGE contains the absolute frequencies of the applied rules by node of the selected derivation trees.

2) A smoothing operator is applied to these frequencies. (18) is applied to the production rules weights $\omega_{r_{n_A}}$ of every nonterminal node n_A. The sum of all the weights $\omega_{x_{n_A}}, \forall x \in R_A$ multiplied by a smoothing rate σ is summed to each $\omega_{r_{n_A}}, r \in R_A$.

$$\omega_{r_{n_A}} = \omega_{r_{n_A}} + \sigma \sum_{x \in R_A} \omega_{x_{n_A}}, \sigma \geq 0 \tag{18}$$

3) The probabilistic model calculation. (19) is applied to all n_A of the CFGE.

$$\rho(r_{n_A}) = \frac{\omega_{x_{n_A}}}{\sum_{x \in R_A} \omega_{x_{n_A}}} \tag{19}$$

Instead of directly generating the offspring using the obtained probabilistic model in (19), an incremental approach is adopted (20) so that the probabilistic model is update according to a learning rate $\alpha \in [0,1]$. This way, the diversity preservation is improved even more. The probability of applying a production rule r_{n_A} in the generation g, $\rho_g(r_{n_A})$, is determined by the probability of applying that same production rule in the previous generation, $\rho_{g-1}(r_{n_A})$, the obtained probabilistic model in (19), $\rho(r_{n_A})$, and the learning rate α.

$$\rho_g(r_{n_A}) = (1 - \alpha)\rho_{g-1}(r_{n_A}) + \alpha\rho(r_{n_A}) \tag{20}$$

To generate each individual, CFGE is traversed from the axiom node to the leave nodes according to the probabilistic model. For every n_A visited, a production rule is selected and applied according to the probabilities $\rho_g(r_{n_A}), \forall r \in R_A$ so that new nodes are visited. This process is recursively executed until the leaves nodes are reached. The resulting traversed path in the CFGE is a new derivation tree of the offspring. During the generation process, if a visited node n_A complies with $\sum_{x \in R_A} \rho(x_{n_A}) = 0$, then that node has been reached thanks to the smoothing method. That is, the probability the parent node $\rho(r_n)$ of the node n_A is $\rho(r_n) > 0$ due to the smoothing method (20). Therefore, n_A has been unlikely visited before

and new search space areas are being explored. In this case, the probability to choose a production rule is obtained from the dynamic stochastic CFG presented for GUPI (12).

Table 12 shows the EDX algorithm. Smoothing rate σ and learning rate α are provided together with a set of parent derivation. This algorithm returns a new set of derivation trees based on parent characteristics.

Table 11: Estimation distribution crossover algorithm.

1) *For each derivation tree D in the parents set:*
    2) *For each derivation in D that have applied a production rule r in a node n.*
        3) *Update the CFGE so that $\omega(r_n) = \omega(r_n) + 1$*
4) *For each node n of the CFGE:*
    5) *Update the weights $\omega(r_n)$ according (18).*
    6) *Obtain $\rho(r_n)$ according to (19)*
7) *Repeat N times*
    8) *Set CFGE root node as current node C.*
    9) *Choose a production rule r following the computed probabilities. If $\sum_{x \in R_A} \rho(x_n) > 0$ then use (19), otherwise, (12).*
    10) *For every nonterminal node n produced by r, repeat from step 9 with $C = n$.*

6.1. Results

Optimization experiments have been conducted to compare the performance of the overall GGGP evolutionary process when using either WX, an incremental EDA or the proposed EDX. Different experiments are accomplished to show the performance of the three approaches when trying to obtain derivation trees with different number of recursive derivations: 5, 10, 20, 50, 100 and 250. For each of these values, a random derivation tree is generated. These are the target derivation trees that defines the problem to solve. An experiment is performed for each of the target derivation trees where each GGGP approach tries to obtain that derivation tree or a similar one. Derivation trees with 0 recursive derivations are not included in the experiments since the initial population usually contains the target derivation tree. The GGGP fitness function is the hamming distance between the word encoded in the target derivation tree and the word of the evaluated derivation tree. The derivations words are compared, so that, for each position of the words, the fitness value is increased by one every time the elements in that position are

different. If the lengths of the words are not the same, the difference of the lengths is summed to the fitness value since the last elements of the longest word cannot be compared. The evolutionary process seeks to minimize the fitness value. The evolutionary process converges if it finds the target derivation tree or it accomplishes 50 generations without improving the average population fitness. The WX chooses 2 individuals to produce 2 new individuals. EDA chooses 50% of the population to produce a new whole generation. The EDA's incremental learning rate is set to 0.3 so that a 0.7 weight is applied to the previous probabilistic model and 0.3 weight to the selected individuals probabilistic model. The EDX chooses 50% of the population to produce an offspring whose size is 25% of the whole population size. The learning rate is also set to 0.3 and the smoothing rate to 0.001. All of them employ the tournament selection to improve the population diversity. No mutation or immigration are applied to reduce the random component of the evolutionary process and facilitate the comparison. Population size is fixed to 100. 50 runs are performed for each experiment with the same target derivation. Comparisons have been applied in terms of number of generations, the average population fitness and the best fitness of the population.

Same as the section 5.2.2, two types of plots are employed to show the results: an evolution of the average population fitness of each GGGP experiment, showing each of the 50 execution runs for each approach, and a statistical study of the best fitness obtained and the number generations for all runs of each experiment. In general, the estimation of distribution approaches show a better performance in terms of fitness and convergence speed than the WX approach. Moreover, the proposed EDX obtains a better average of the best fitness than EDA and WX approaches in most of the experiments independently of the CFG or the number of recursive derivations of the target derivation tree.

6.1.1. Optimization Experiments

Feed-forward Neural Network (1 hidden layer) Optimization

The performed experiments with the G_{FFNN} shows a faster improvement of the average fitness evolution of both estimation of distribution approaches compared to the WX approach independently of the target derivation tree (Figure 132 to Figure 137). The EDA approach shows even a slightly faster improvement of the average fitness in the first iterations than the EDX approach since it produces a new better adapted population every generation. However, the EDX approach finally outperforms both in terms of fitness and it obtains better final results independently of the target derivation tree since it provides an enhanced diversity preservation (Figure 138). The proposed model smoothing increases the probability of exploring new areas of the search space increasing this way the probability to find new better adapted individuals. Statistical evidences are provided in Figure 138 to state there is a statistically significant difference between the fitness means ($p < 0.05$) independently of the number of recursive derivations of the target derivation tree. Moreover, Figure 139 shows how EDA and EDX outperforms WX in terms of generations excepting for the cases where WX prematurely converges.

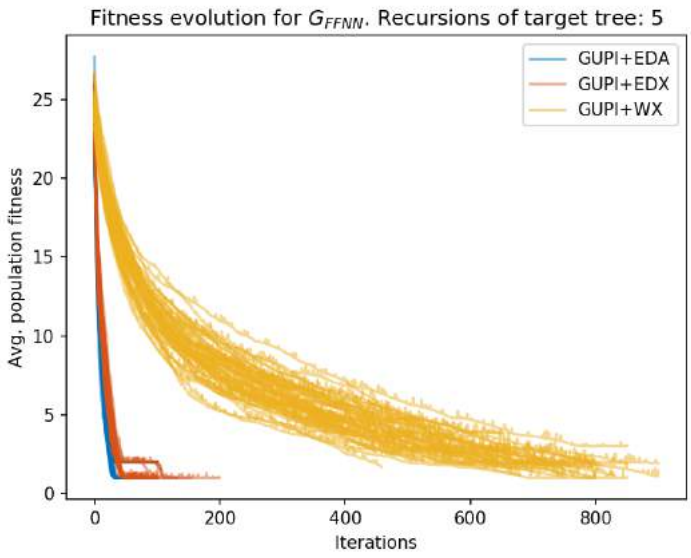

Figure 132: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{FFNN} with 5 recursive derivations is searched. 50 executions run.

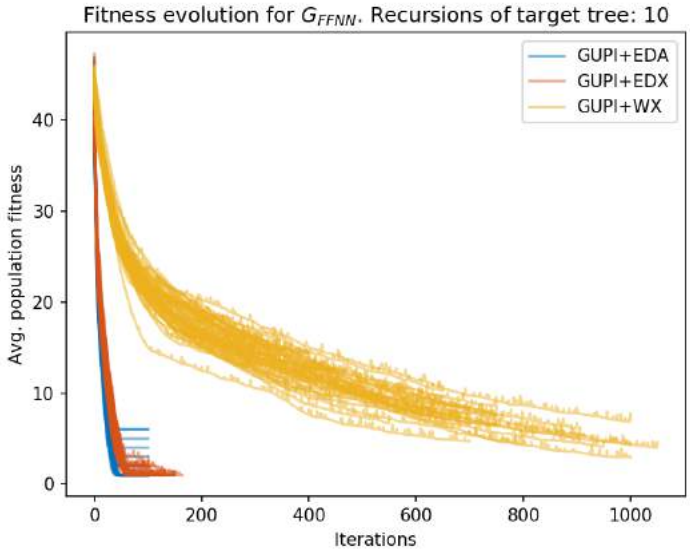

Figure 133: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{FFNN} with 10 recursive derivations is searched. 50 executions run.

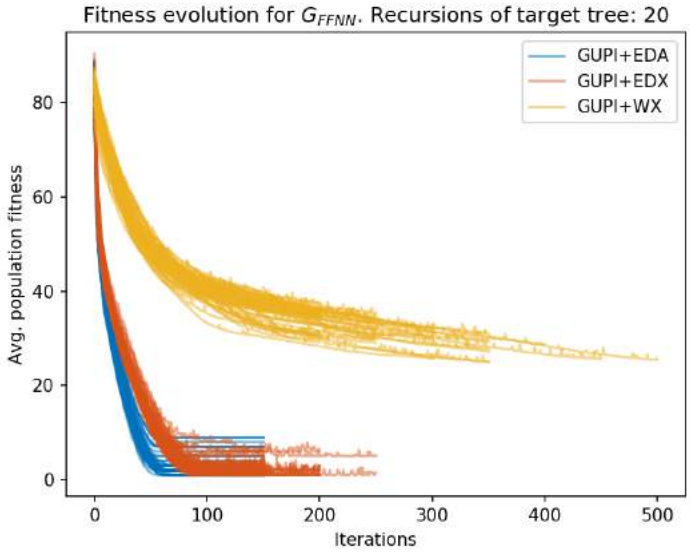

Figure 134: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{FFNN} with 20 recursive derivations is searched.

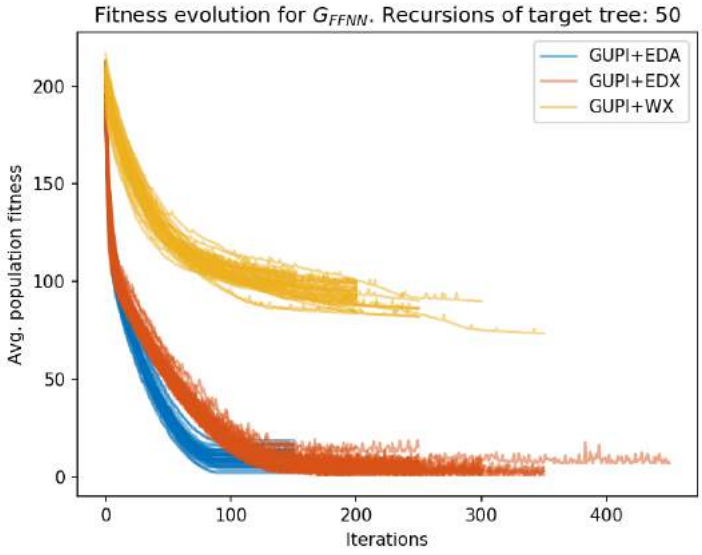

Figure 135: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{FFNN} with 50 recursive derivations is searched. 50 executions run.

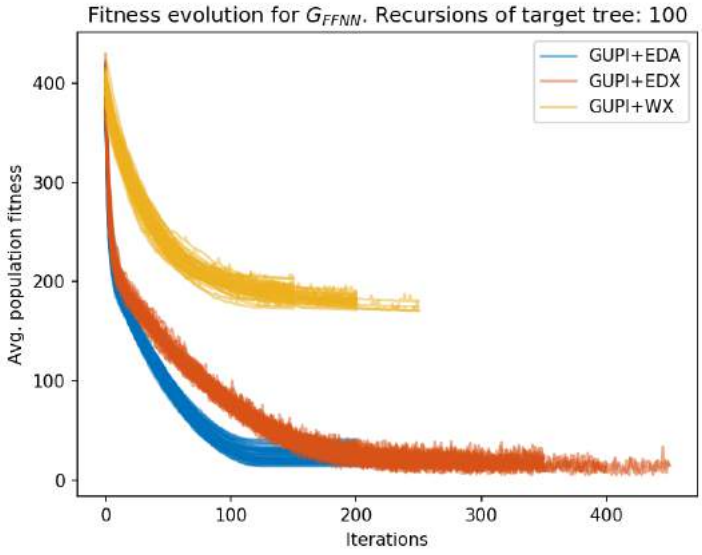

Figure 136: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{FFNN} CFG with 100 recursive derivations is searched. 50 executions run.

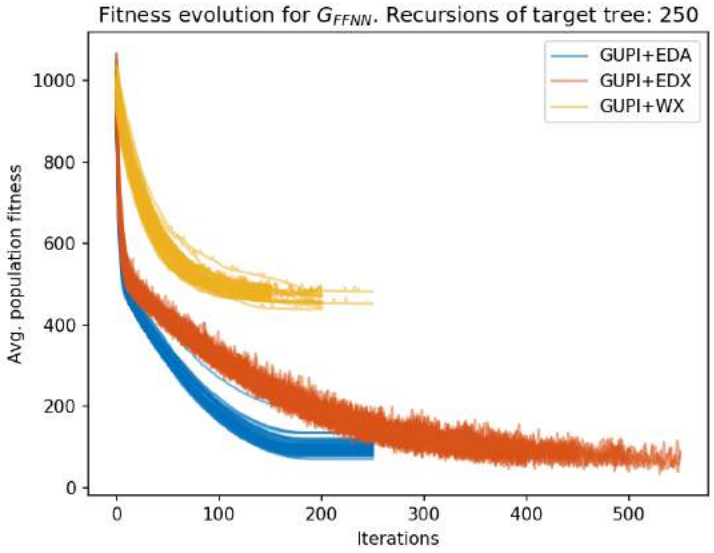

Figure 137: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{FFNN} with 250 recursive derivations is searched. 50 executions run.

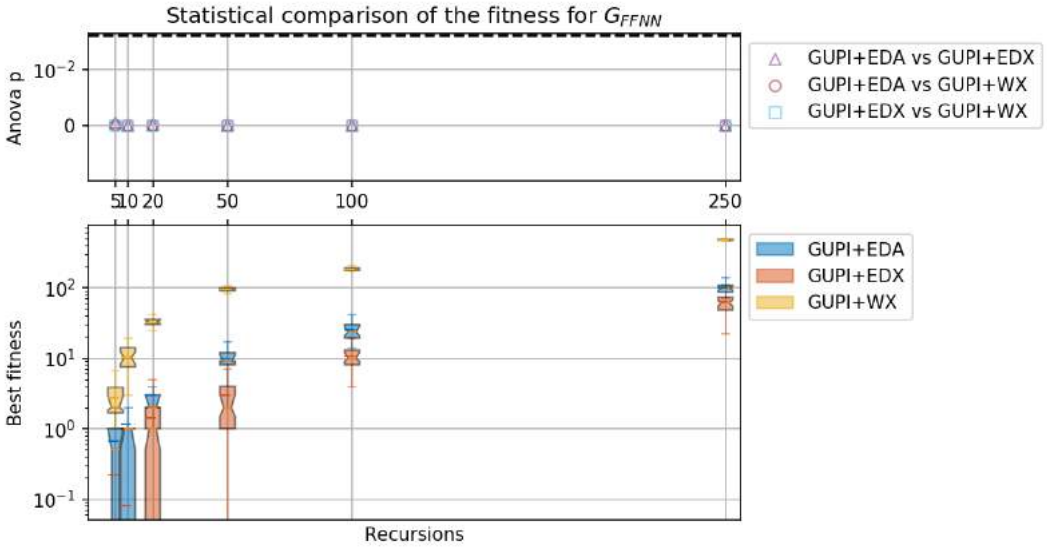

Figure 138: One-way ANOVA of the average fitness for G_{FFNN} to compare EDX with an incremental EDA approach. 50 executions run per target derivation tree.

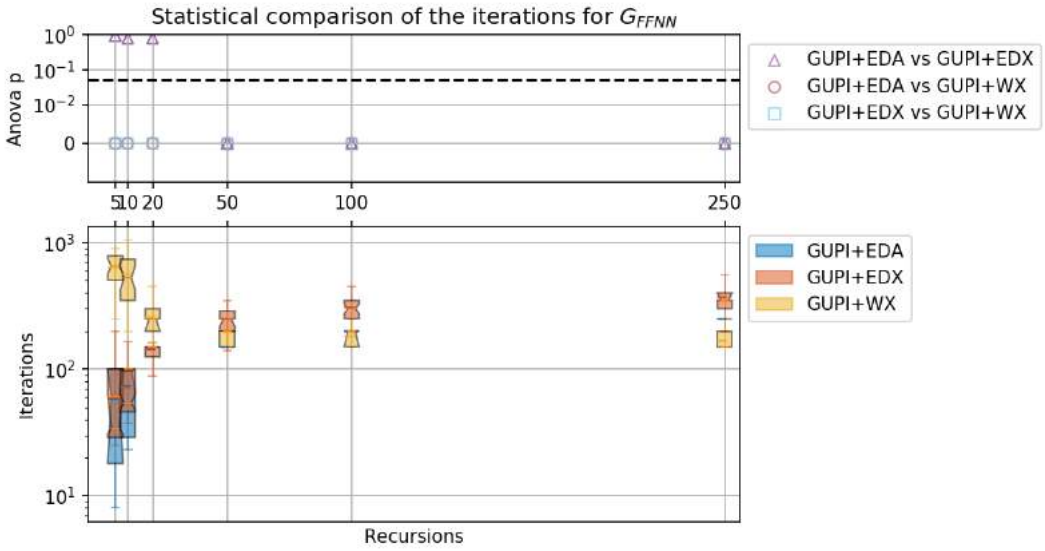

Figure 139: One-way ANOVA of the average iterations for G_{FFNN} to compare EDX with an incremental EDA approach. 50 executions run per target derivation tree.

Deep Feed-forward Neural Network Optimization

The performed experiments with the G_{DFFNN} also shows a faster improvement of the average fitness evolution of the estimation of distribution approaches compared to the WX approach, independently of the target derivation tree (Figure 140 to Figure 145). The EDA and EDX approaches shows a very similar performance during the whole evolutionary process. Both shows some difficulties to improve the population fitness when the population has already converged to a promising solution (Figure 142). However, the EDX approach manages to explore new search space areas and obtain better adapted individuals. In particular, Figure 142 shows how EDX is able to explore and obtain better adapted individuals when the population has already converged. The EDX approach outperforms WX in terms of fitness and it obtains better final results independently of the target derivation tree (Figure 146). Moreover, EDX approach also outperforms the EDA approach in terms of fitness when searching for derivation trees with 5, 20, 100, 250. Statistical evidences are provided in Figure 146 to state there is a statistically significant difference between the fitness means ($p < 0.05$) when searching for these target derivation trees. For the target derivation trees with 10 and 20 recursive derivations, no statistically significant differences are obtained.

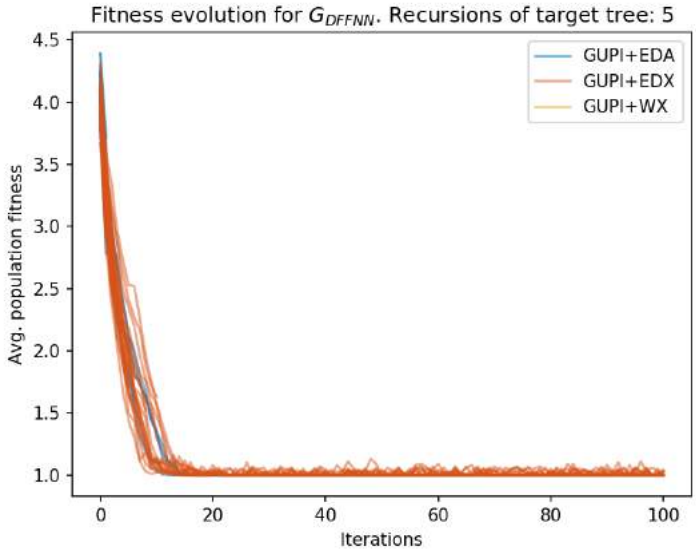

Figure 140: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{DFFNN} with 5 recursive derivations is searched. 50 executions run.

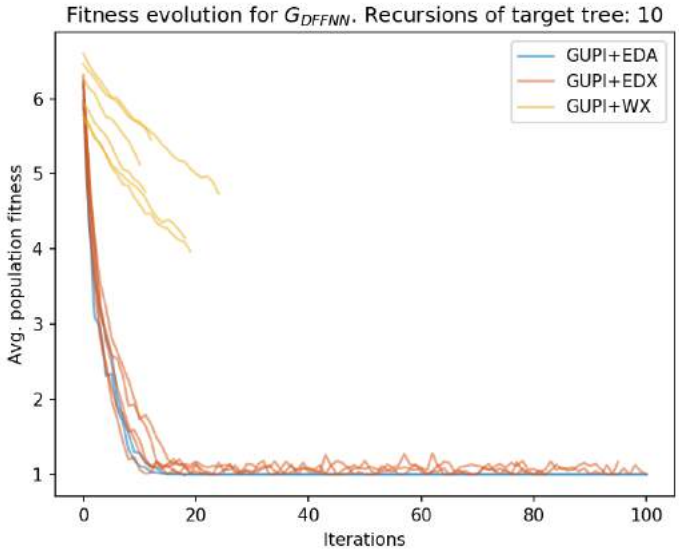

Figure 141: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{DFFNN} with 10 recursive derivations is searched. 50 executions run.

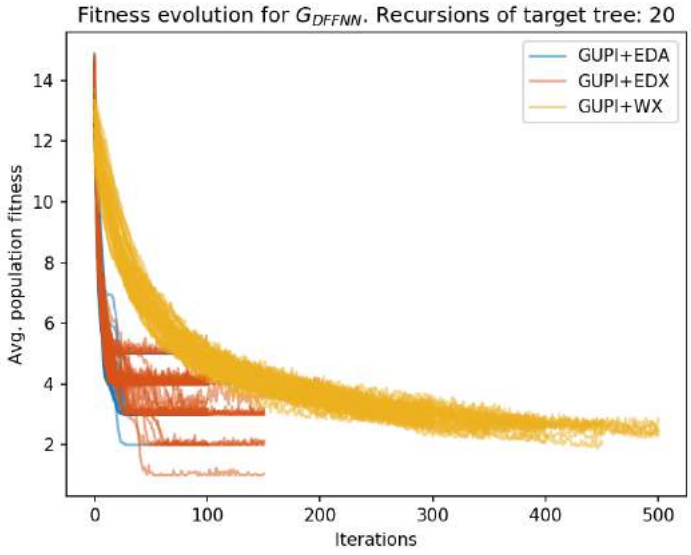

Figure 142: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{DFFNN} with 20 recursive derivations is searched. 50 executions run.

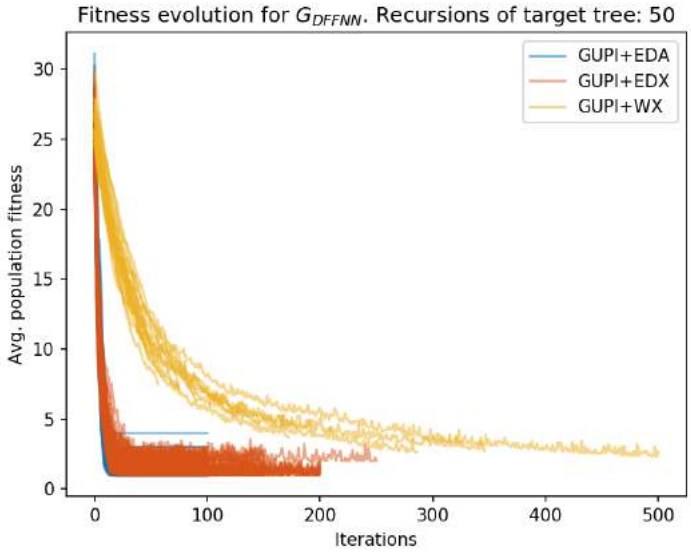

Figure 143: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{DFFNN} with 50 recursive derivations is searched. 50 executions run.

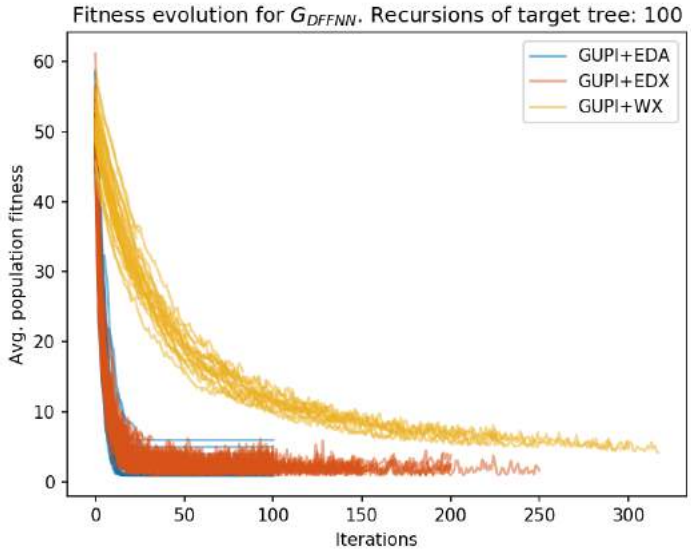

Figure 144: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{DFFNN} with 100 recursive derivations is searched. 50 executions run.

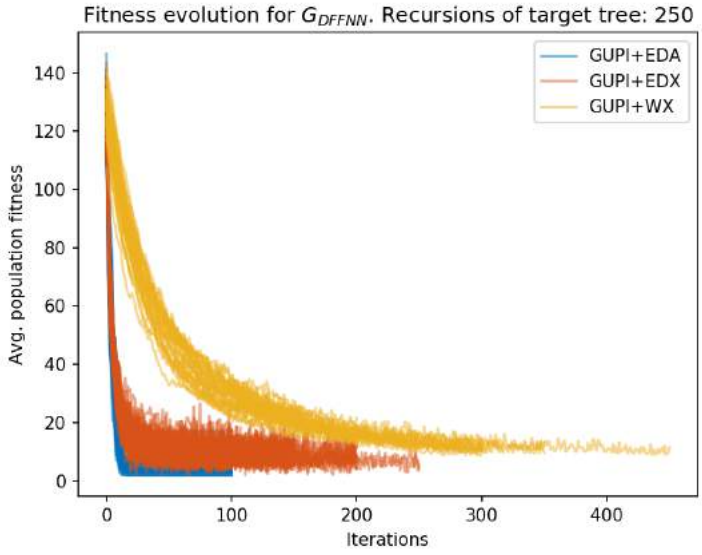

Figure 145: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{DFFNN} with 250 recursive derivations is searched. 50 executions run.

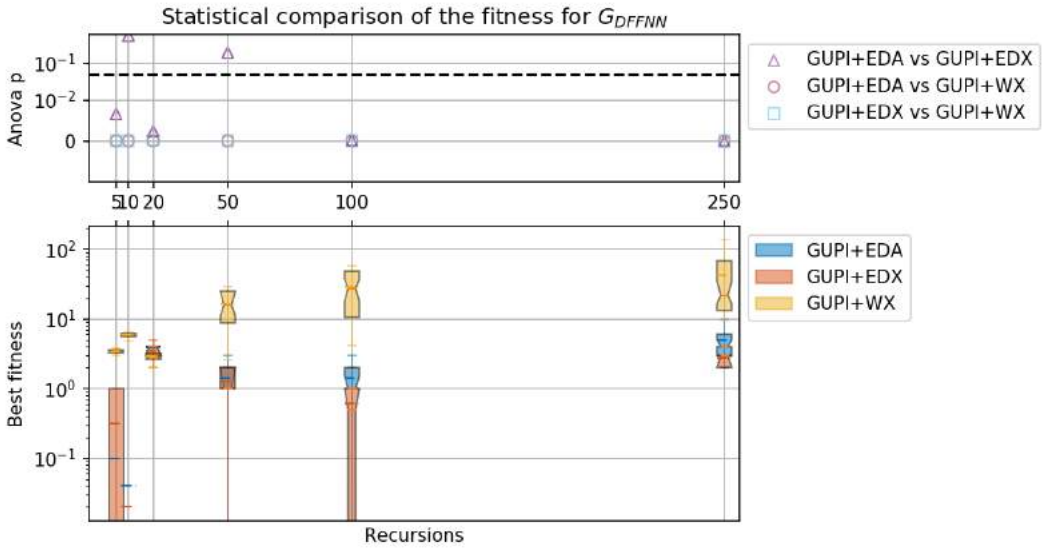

Figure 146: One-way ANOVA of the average fitness for G_{DFFNN} to compare EDX with an incremental EDA approach. 50 executions run per target derivation tree.

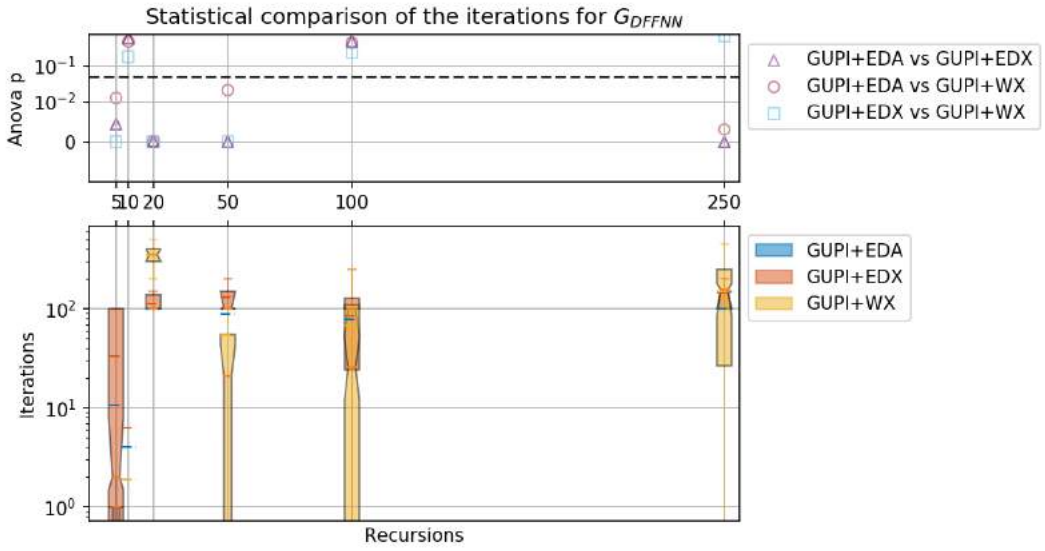

Figure 147: One-way ANOVA of the average iterations for G_{DFFNN} to compare EDX with an incremental EDA approach. 50 executions run per target derivation tree.

Rule-Based System Optimization

The performed experiments with the G_{RBS} also shows a faster improvement of the average fitness evolution of the estimation of distribution approaches compared to the WX approach independently of the target derivation tree (Figure 148 to Figure 153). The EDA approach also shows a slightly faster improvement of the average fitness than the EDX approach when searching for large derivation trees (Figure

150 to Figure 153). However, the EDX approach finally outperforms both and it obtains lower fitness values independently of the target derivation tree (Figure 154). The EDA approach shows difficulties to improve the population fitness when the population has already converged to a promising solution and in general it shows premature convergence (Figure 148 to Figure 153). On the contrary, the EDX approach manages to continue the optimization process thanks to the enhanced diversity preservation and the exploratory capabilities. Statistical evidences are provided in Figure 154 to state there is a statistically significant difference between the fitness means ($p < 0.05$) independently of the number of derivations of the target derivation tree. Moreover, Figure 155 shows how EDA and EDX outperforms WX in terms of generations excepting for the cases where WX prematurely converges.

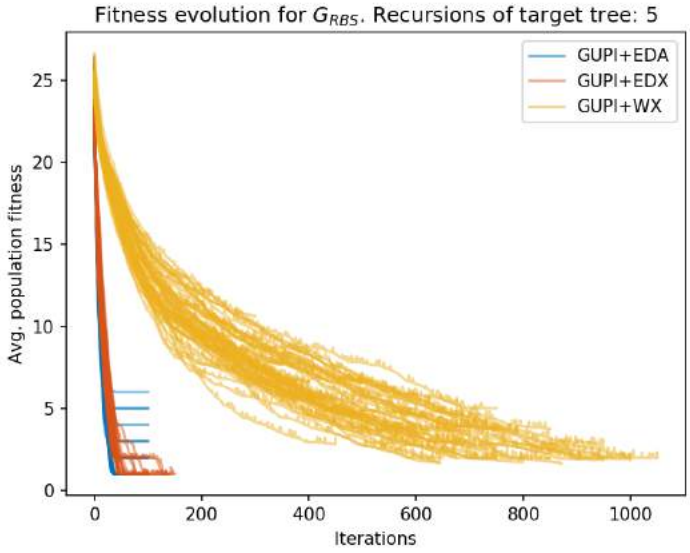

Figure 148: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{RBS} with 5 recursive derivations is searched. 50 executions run.

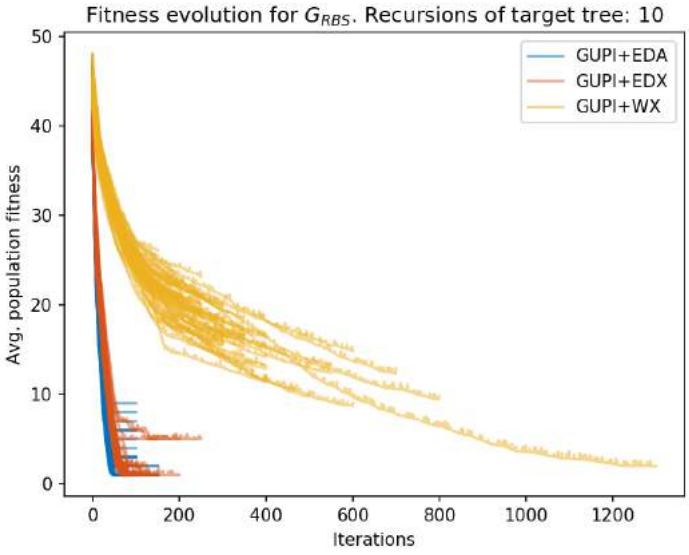

Figure 149: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{RBS} with 10 recursive derivations is searched. 50 executions run.

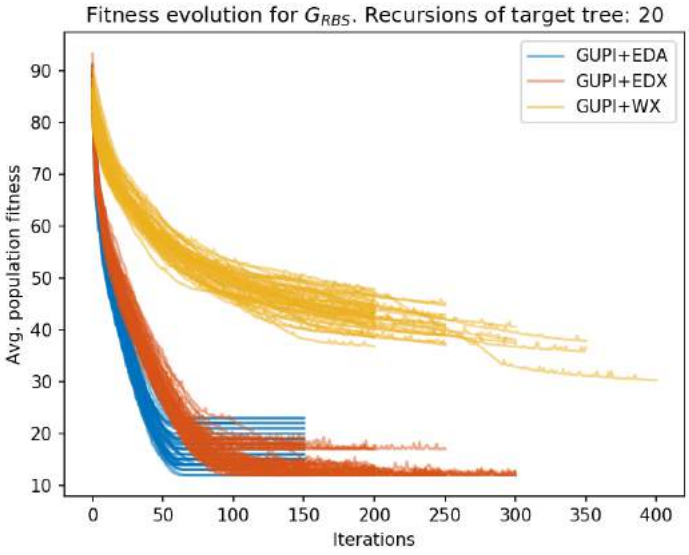

Figure 150: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{RBS} with 20 recursive derivations is searched. 50 executions run.

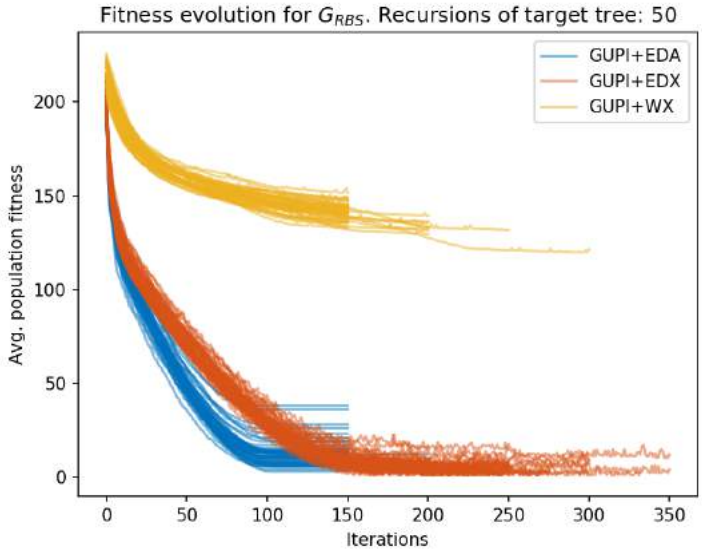

Figure 151: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{RBS} with 50 recursive derivations is searched. 50 executions run.

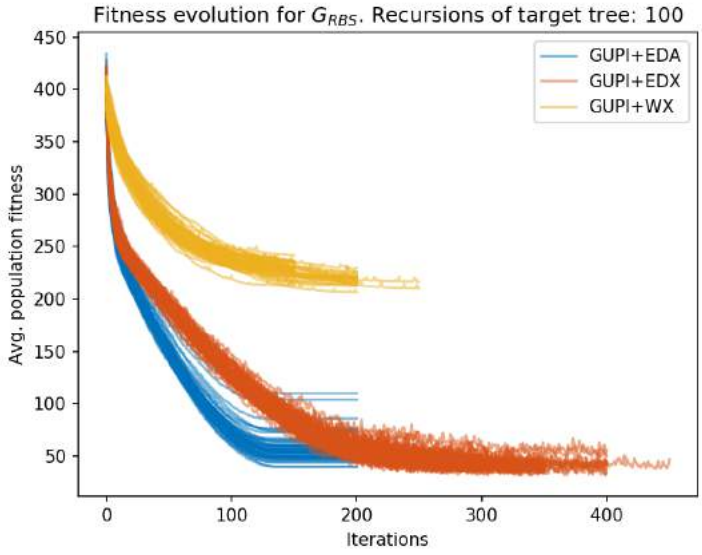

Figure 152: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{RBS} with 100 recursive derivations is searched. 50 executions run.

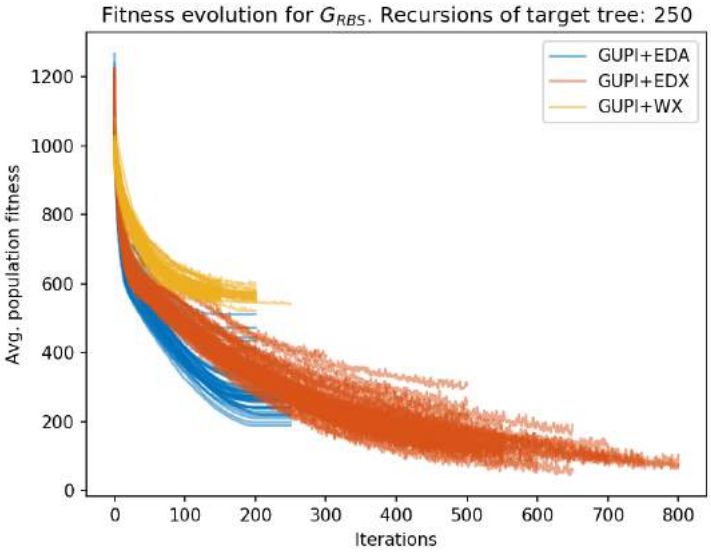

Figure 153: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{RBS} with 250 recursive derivations is searched. 50 executions run.

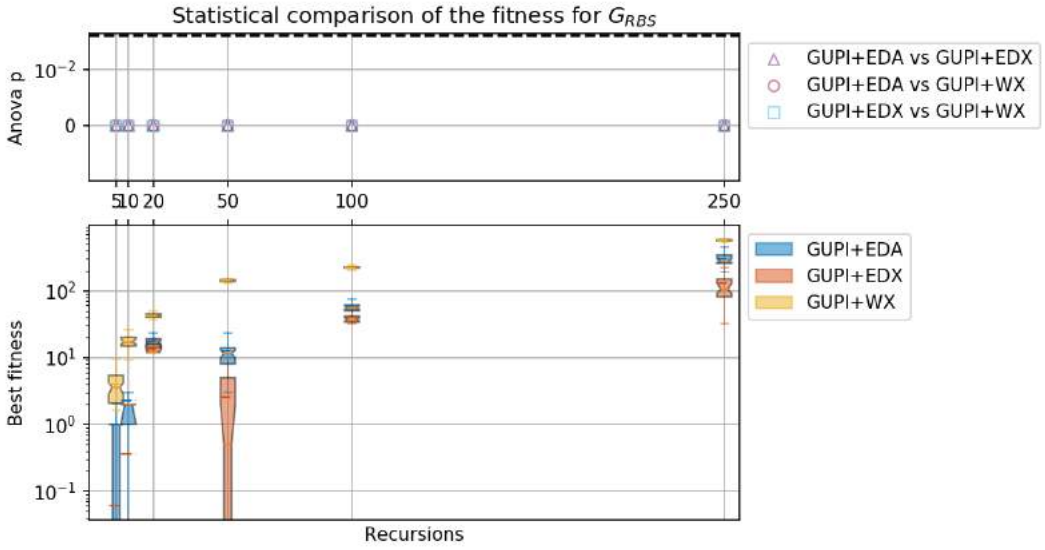

Figure 154: One-way ANOVA of the average fitness for G_{RBS} to compare EDX with an incremental EDA approach. 50 executions run per target derivation tree.

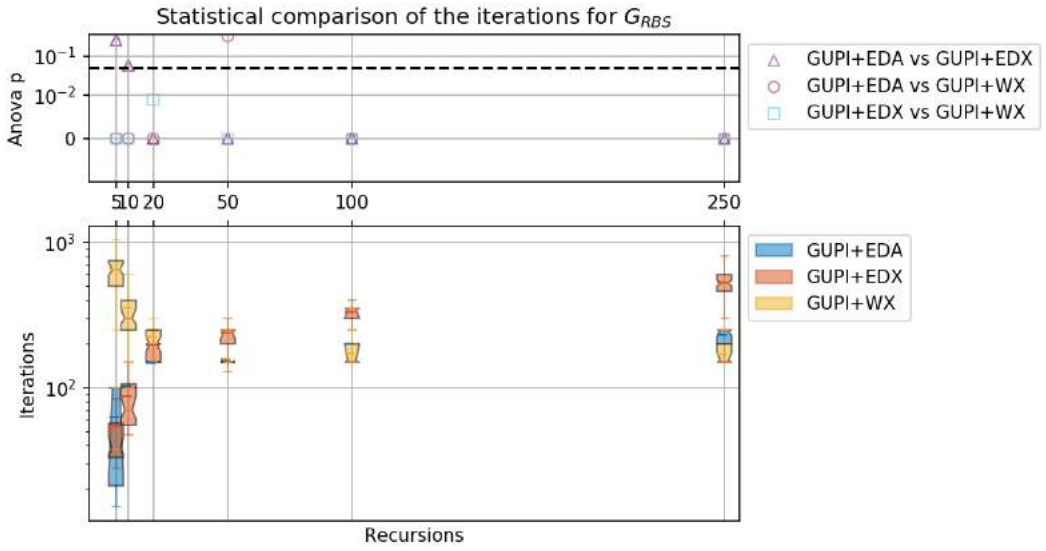

Figure 155: One-way ANOVA of the average iterations for G_{RBS} to compare EDX with an incremental EDA approach. 50 executions run per target derivation tree.

Symbolic Regression Optimization

The performed experiments with the G_{SR} shows a similar pattern in the evolution of the average population fitness independently of the approach (Figure 156 to Figure 161). However, The EDA and EDX approaches outperform WX in terms of fitness and they usually show lower average fitness during the whole evolutionary process. The EDA and EDX approaches show similar difficulties to improve the population fitness when the population has already converged to a promising solution when searching for derivation trees with 5, 10 and 20 recursive derivations (Figure 156 to Figure 158). However, the EDX approach manages to explore new search space areas and, in particular, Figure 157 and Figure 158 shows how EDX significantly improves the average population fitness. When searching for derivation trees with 50, 100 and 250 recursive derivations, EDX shows a better performance than EDA which rapidly converge (Figure 159 to Figure 161). The EDX approach outperforms EDA and WX approaches in the presented experiments (Figure 162). Moreover, statistical evidences are provided to state there is a statistically significant difference between the fitness means ($p < 0.05$) independently of the number of derivations of the target derivation tree (Figure

162). Moreover, Figure 163 provides evidences of the premature convergence of the EDA approach in most of the experiments.

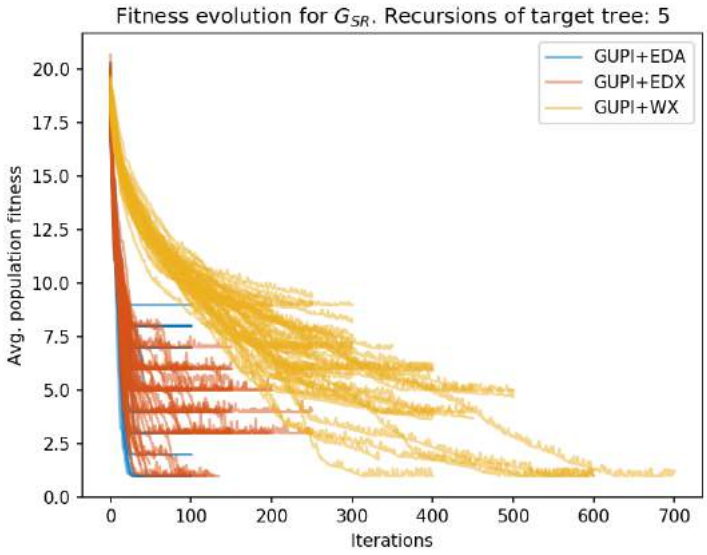

Figure 156: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{SR} with 5 recursive derivations is searched. 50 executions run.

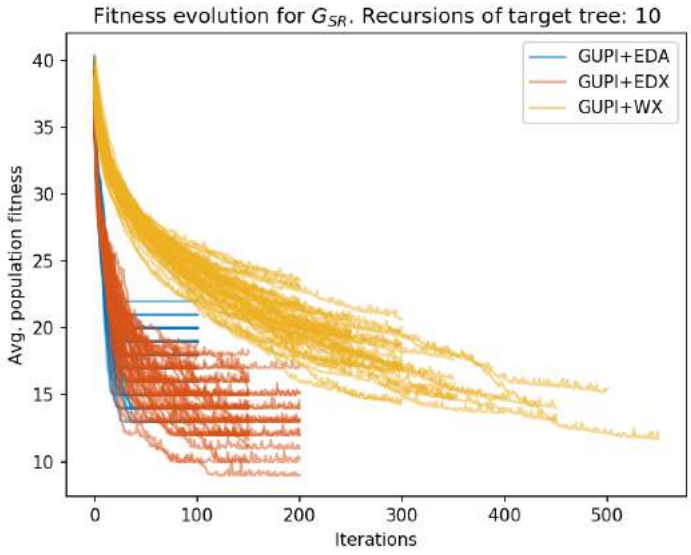

Figure 157: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{SR} with 10 recursive derivations is searched. 50 executions run.

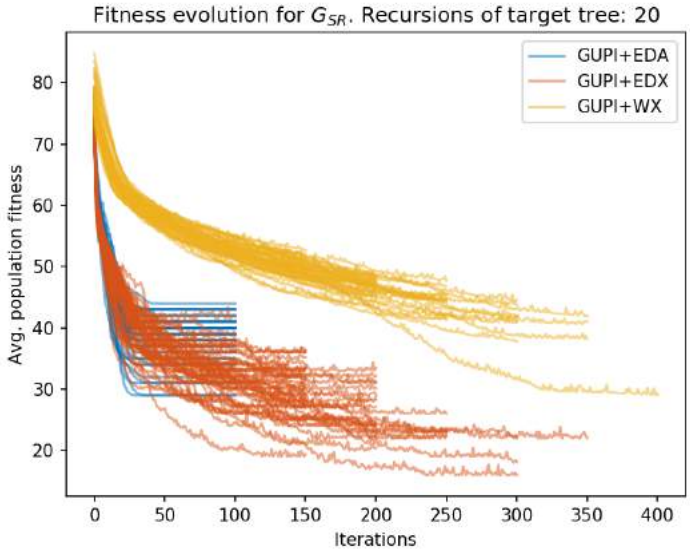

Figure 158: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{SR} with 20 recursive derivations is searched. 50 executions run.

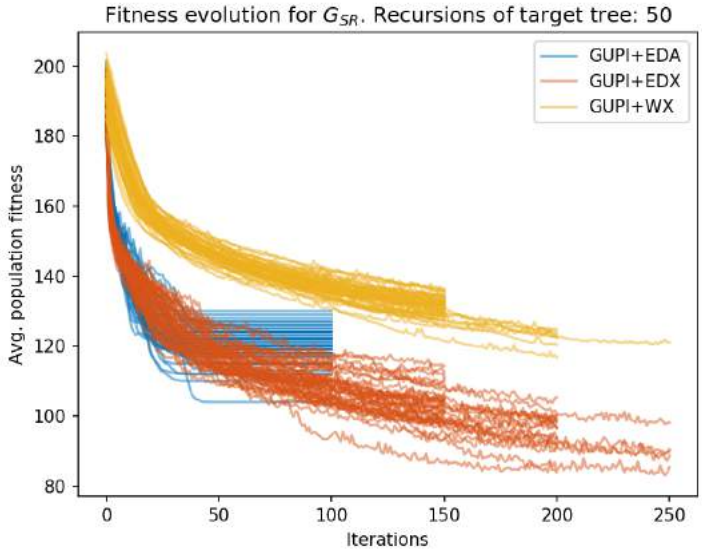

Figure 159: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{SR} with 50 recursive derivations is searched. 50 executions run.

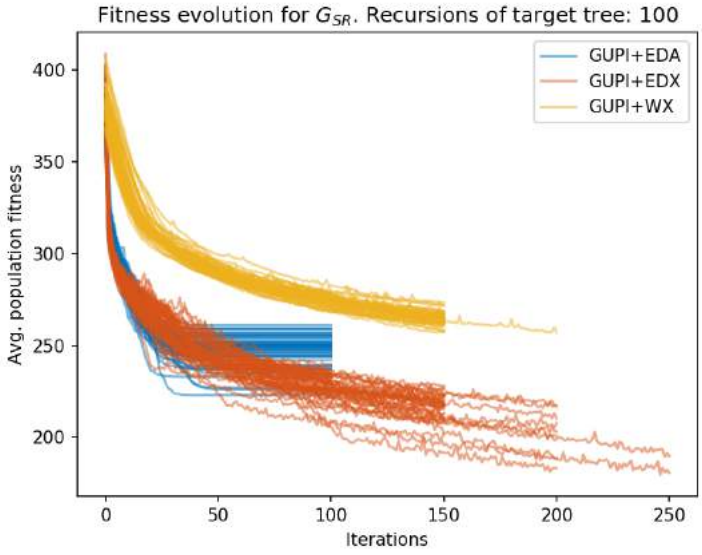

Figure 160: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{SR} with 100 recursive derivations is searched. 50 executions run.

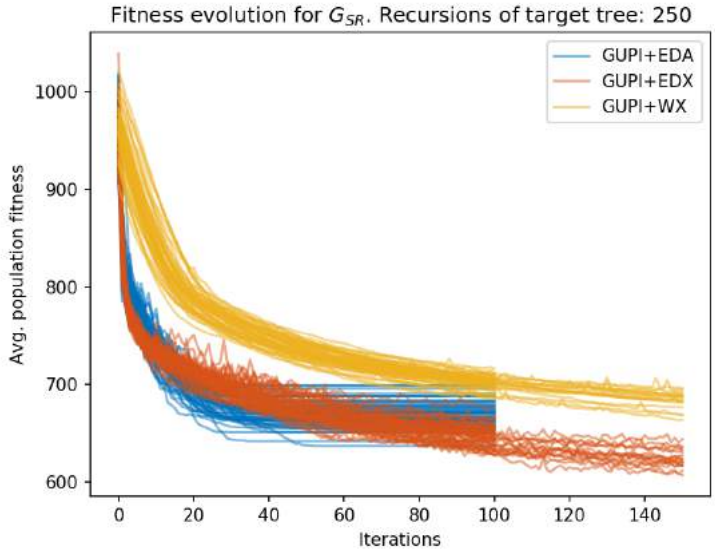

Figure 161: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{SR} with 250 recursive derivations is searched. 50 executions run.

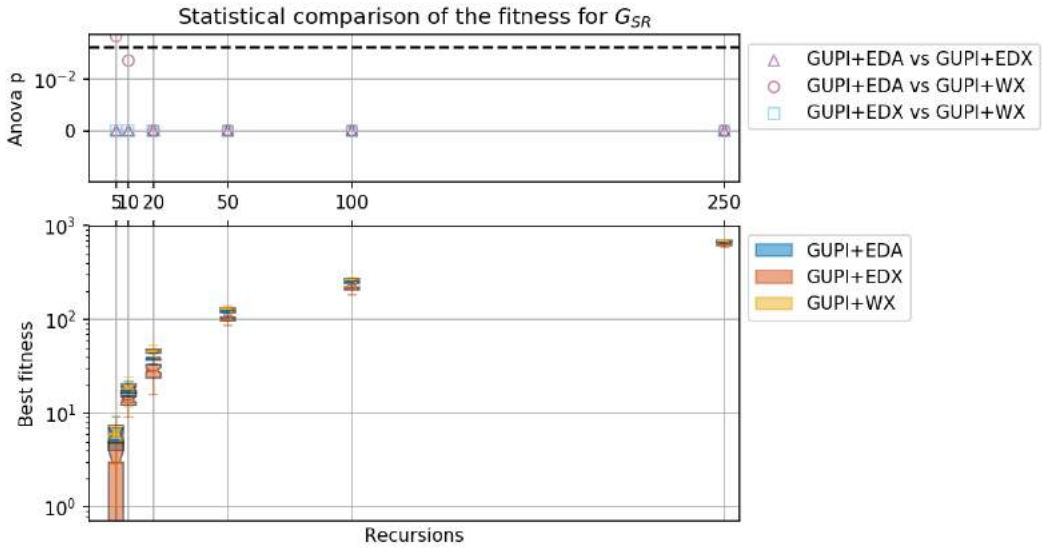

Figure 162: One-way ANOVA of the average fitness for G_{SR} to compare EDX with an incremental EDA approach. 50 executions run per target derivation tree.

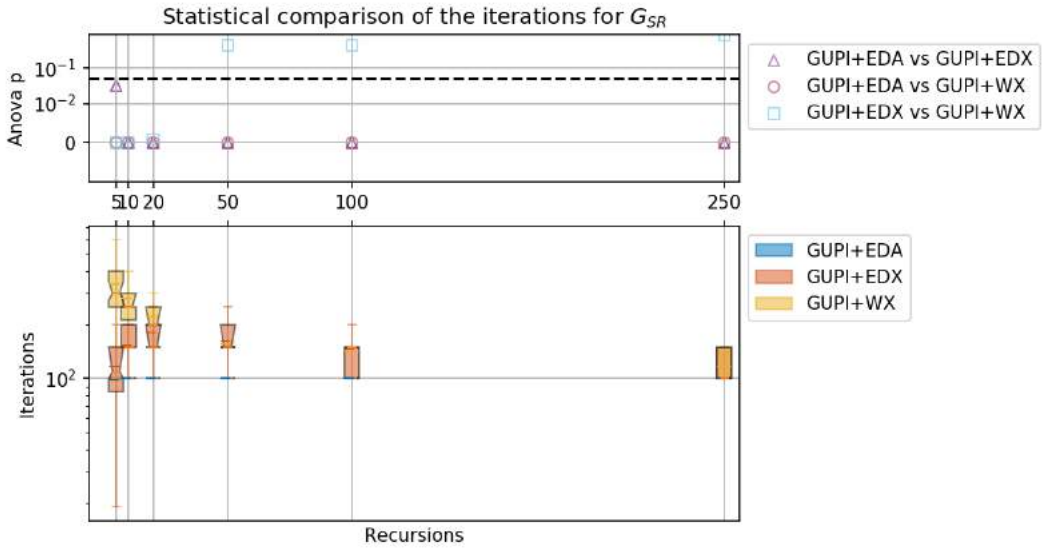

Figure 163: One-way ANOVA of the average iterations for G_{SR} to compare EDX with an incremental EDA approach. 50 executions run per target derivation tree.

Boolean Optimization Problem

Very similar results to G_{SR} experiments are obtained in the performed experiments with the G_{BP}. The EDX approach outperforms EDA and WX approaches in terms of fitness in the presented experiments (Figure 170). Statistical evidences are provided to state there is a statistically significant difference between the fitness means ($p < 0.05$) independently of the number of derivations of the target

derivation tree (Figure 170). Figure 166 to Figure 169 show the EDX capabilities to preserve the population diversity and to explore new search space areas so that the premature convergence is avoided. In particular, Figure 164 shows effective exploration operations that rapidly improves the average population fitness from values close to 6 to values close to 1. Moreover, Figure 171 provides evidences of the premature convergence of the EDA approach in most of the experiments.

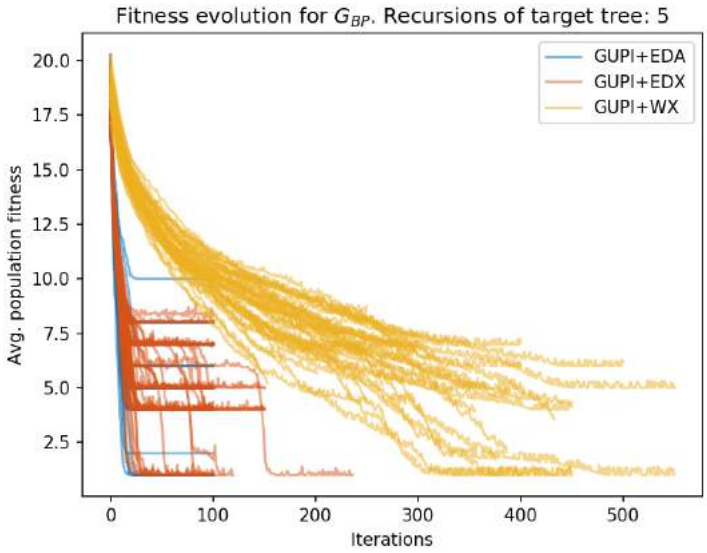

Figure 164: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{BP} with 5 recursive derivations is searched. 50 executions run.

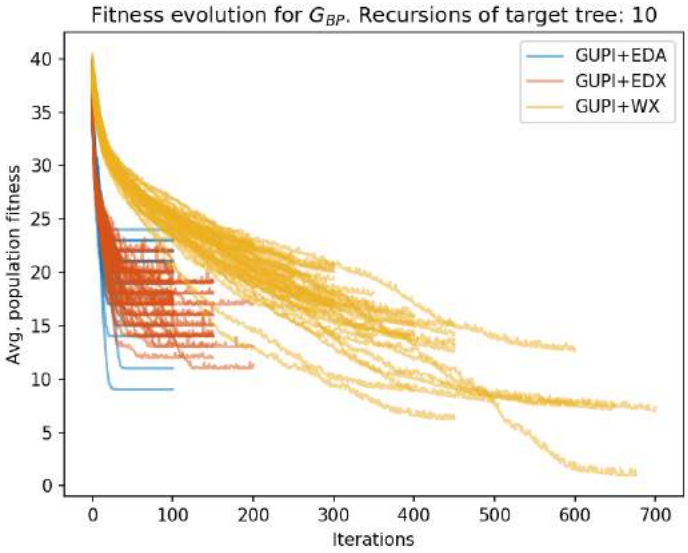

Figure 165: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{BP} with 10 recursive derivations is searched. 50 executions run.

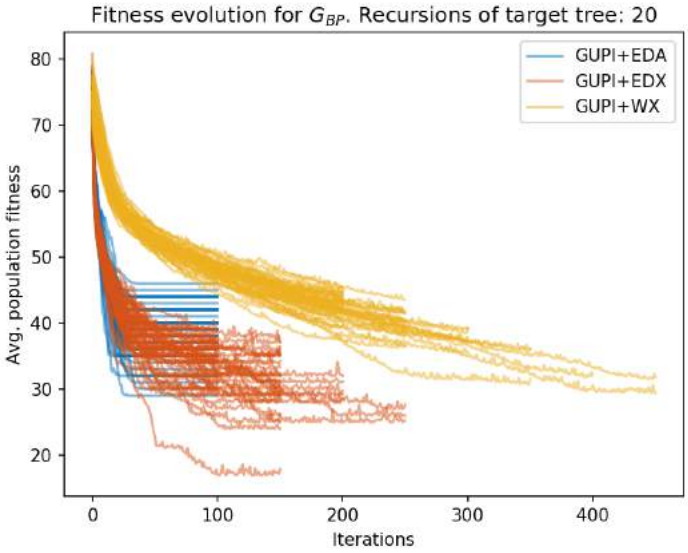

Figure 166: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{BP} with 20 recursive derivations is searched. 50 executions run.

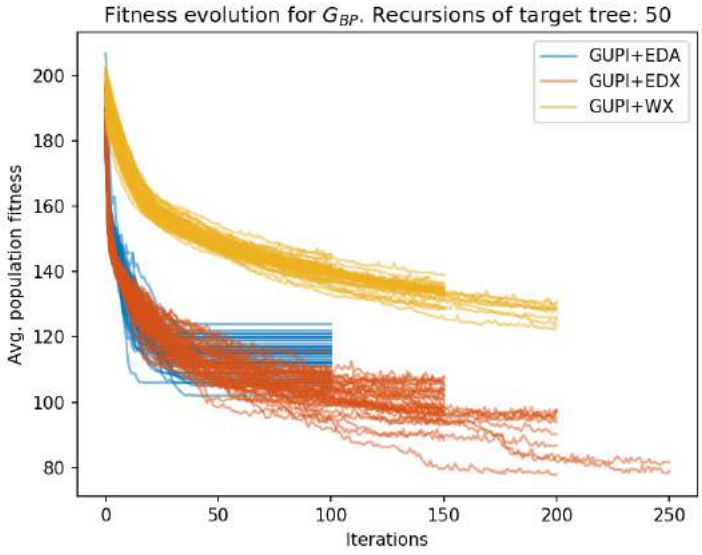

Figure 167: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{BP} with 50 recursive derivations is searched. 50 executions run.

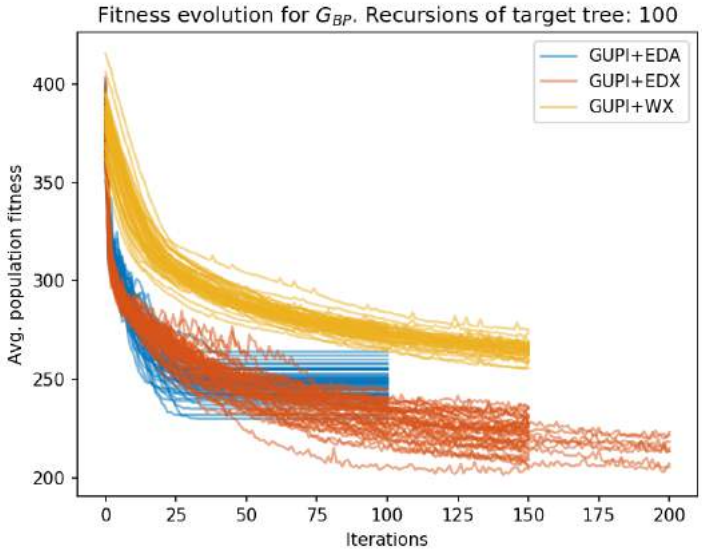

Figure 168: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{BP} with 100 recursive derivations is searched. 50 executions run.

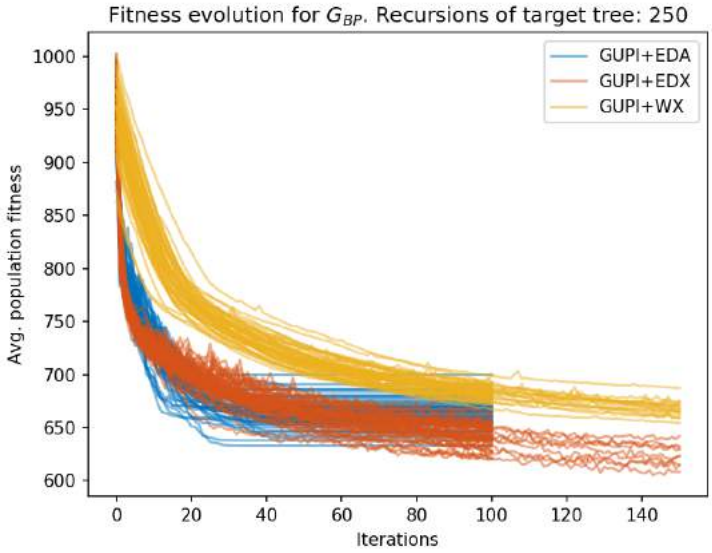

Figure 169: Fitness evolution comparison of GGGP optimization when using EDX, an incremental EDA or WX with GUPI. A derivation tree of G_{BP} with 250 recursive derivations is searched. 50 executions run.

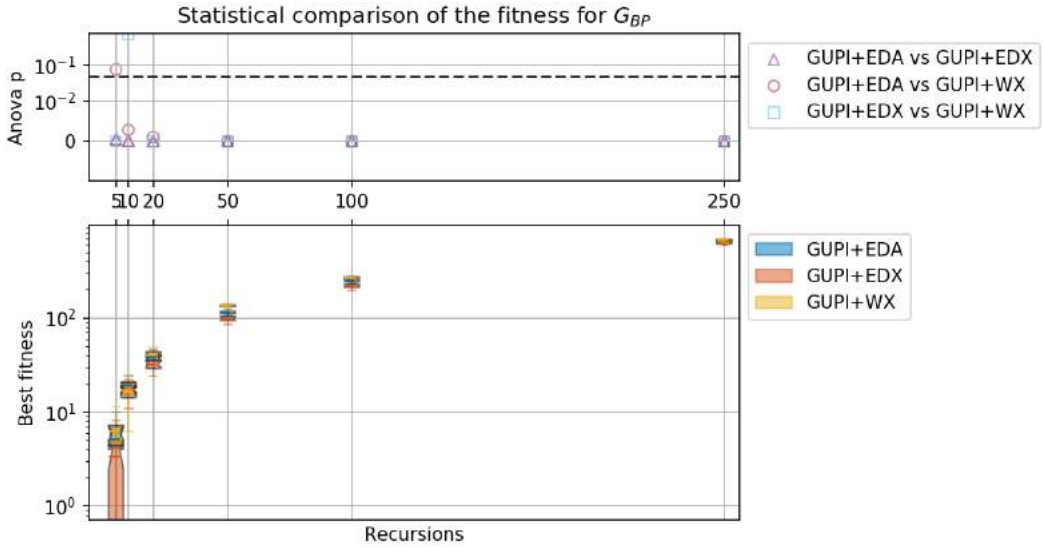

Figure 170: One-way ANOVA of the average fitness for G_{BP} to compare EDX with an incremental EDA approach. 50 executions run per target derivation tree.

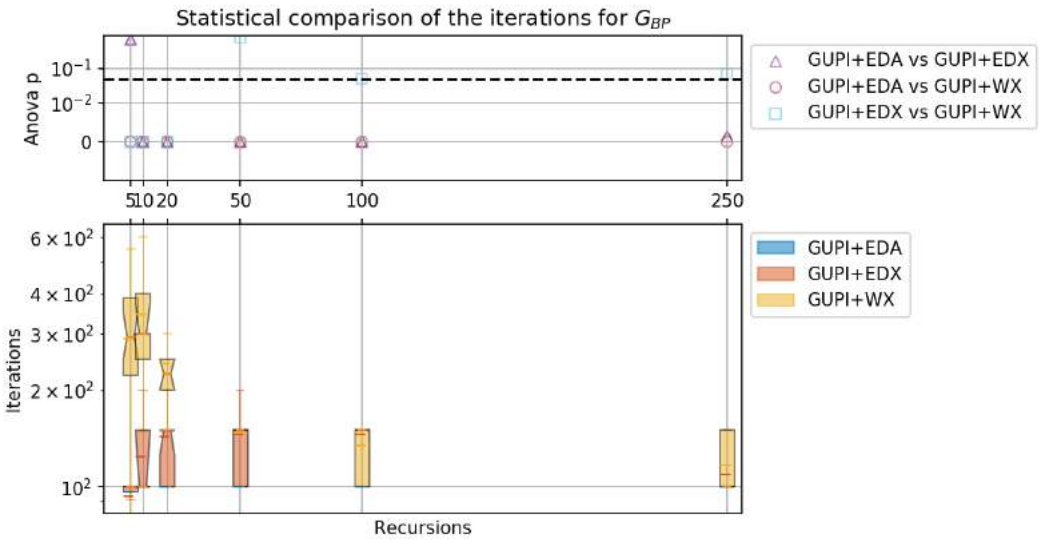

Figure 171: One-way ANOVA of the average iterations for G_{BP} to compare EDX with an incremental EDA approach. 50 executions run per target derivation tree.

7. Endosymbiotic Co-Optimization System

Endosymbiotic co-optimization is a novel optimization and search technique that overcomes EC limitations when dealing with variable size solutions with high cardinality terminal symbols sets. To such aim, ECO takes advantage of symbiosis and phenotypic plasticity benefits. ECO comprises two different kind of optimization: the evolution and the learning processes. The evolution process defines the structure of ECO individuals: possible encoded solutions for the problem at hand. Each individual possesses learning capacities that improves its encoded solution. The learning process is concurrently and asynchronously executed during the whole individual's life. The individual's fitness is then continuously updated according to its learning improvements. These asynchronous updates provide estimates of individual goodness that guides the evolution process. Therefore, the originality of ECO lies in the way the evolution and the learning optimization techniques interact, not in the optimization and search techniques themselves.

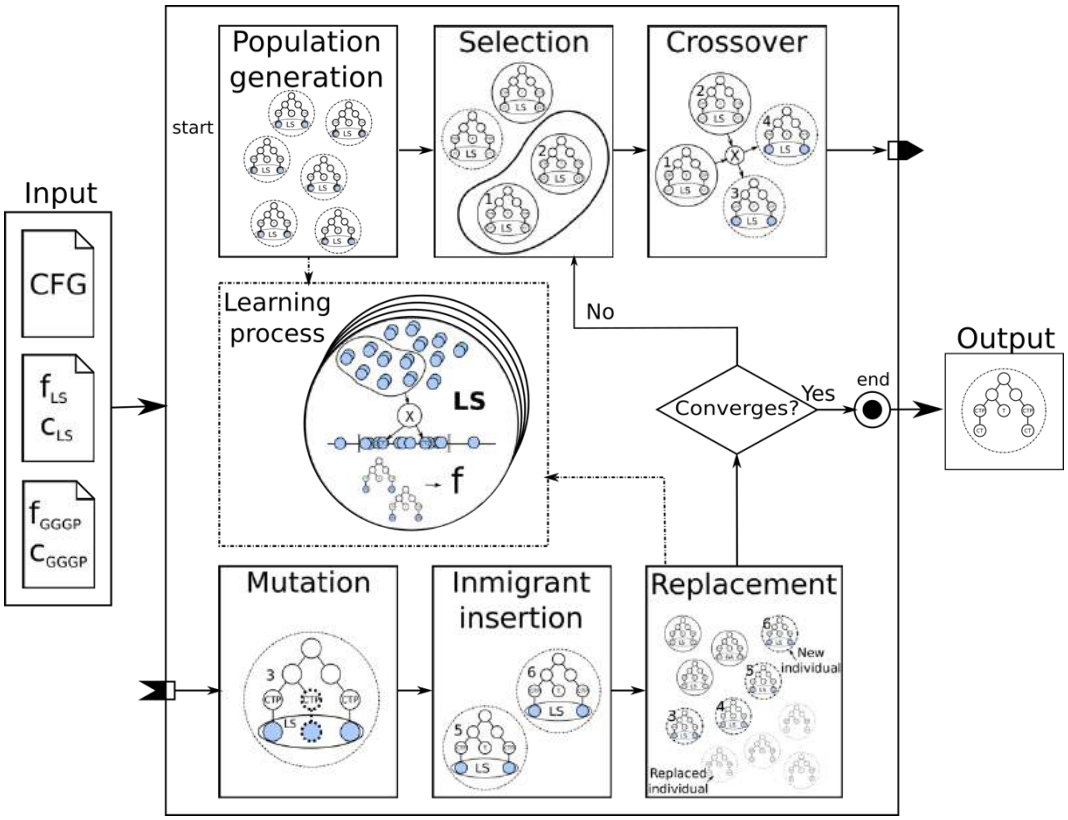

Figure 172: Endosymbiotic co-evolutionary overview

Figure 172 shows an overview of the ECO system. A GGGP algorithm performs the evolution optimization process. It implements the common evolutionary steps: population generation, selection, crossover, mutation, immigrant insertion and replacement. Selection operator employs tournament method. Then, crossover and mutation act with certain probability to generate the offspring. A set of immigrant individuals are newly created and inserted in the population to introduce genetic diversity in changing environments (Cobb & Grefenstette, 1993). The replacement operator chooses the worst individuals of the population to be replaced by the offspring and the immigrant population. Each ECO individual is a derivation tree that encodes a set of production rules that generates a word belonging to the CFG, i.e. a feasible solution for the problem at hand.

The CFG is defined as a 5-tuple $G = (V, \sum, R, CTR, S)$, where V is the alphabet of nonterminal symbols, $\sum$ is the alphabet of terminal symbols and $V \cap \sum = \emptyset$. $S \in V$ is the axiom or root. R and CTR are the sets of production rules, where $R \cap CTR = \emptyset$. R and CTR are written in Backus-Naur form. Those production rules whose

consequents only involve terminal symbols are called terminal productions. That is, $r: A ::= \alpha, A \in V_0$. There is a subset of those terminal productions that are especially critical because they decrease the GGGP performance. They are the terminal productions of the CFG whose number of terminals associated is high, e.g. terminal productions that derivate into natural numbers. This subset of terminal production rules is called critical terminal productions (CTR) and the terminals produced by them are called critical terminals (CT). R is the remaining set of production rules that belongs to the CFG.

Figure 173 also shows that ECO system receives as inputs the CFG, the fitness functions (f_{ECE}, $f_{Learning}$) and the ECO convergence function (c_{ECE}). The CFG defines the problem search space and the syntactical restrictions that solutions (derivation trees) must comply with. f_{ECE} and $f_{Learning}$ are employed to measure the fitness of the evolutionary individuals. Once the system receives the inputs, ECO system starts and it continues until the stop condition is met(c_{ECE}): maximum number of ECO iterations has been reached, the ECO population converges, or best fitness of the ECO population reaches a pre-established threshold. Then, the best individual of the ECO population is returned.

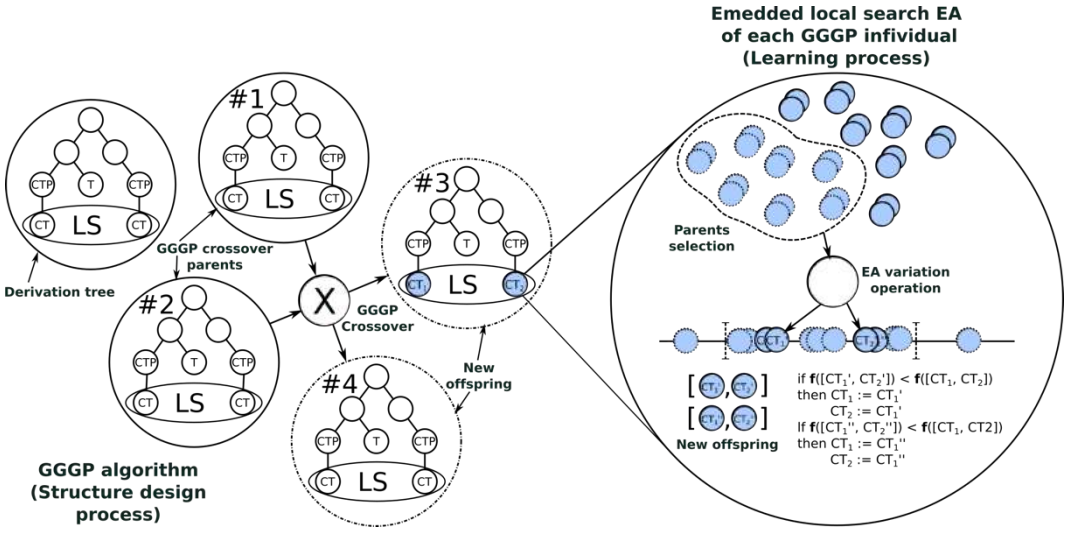

Figure 173: Learning process

Each ECO individual implement its own embedded concurrent learning process. Figure 173 shows the detail of the embedded local search (LS) learning process.

Learning gives the opportunity to ECO individuals to take the most of their genotype (derivation tree). This process asynchronously feeds ECO individuals with critical terminal symbols that guide ECO evolution. When each new ECO individual is created (population initialization, offspring of a crossover operator, mutation, or immigrant individuals), its embedded LS starts the learning process of its critical terminals and it continues until the individual is eliminated from the population or when a stop condition is met. This learning process defines and updates the critical terminals for that ECO individual. The population of the learning process consists of vectors that represent possible combinations of critical terminals for the derivation tree of the ECO individual. Every time a LS learns a new set of critical terminals that improves the current best, the derivation tree is asynchronously updated. Therefore, the ECO individual must be reevaluated throughout the fitness function f_{ECE}.

The fitness function f_{ECE} is defined as:

$$f_{ECO} = \omega_{Learning} f_{Learning} + \sum_{i=1}^{n} \omega_i \varphi_i, \quad \forall \omega_i \mid \omega_{Learning} \geq 0, \omega_{Learning} + \sum_{i=1}^{n} \omega_i = 1, \tag{21}$$

where $f_{Learning}$ is the LS learning process fitness function that shows how suitable a set of critical terminals is for a specific derivation tree. $\omega_{Learning}$ is a weight that represents the relative importance of $f_{Learning}$ in f_{ECO}. φ_i are additional features that the solution must accomplish, i.e. size of the solution or a measure of its complexity. These additional features are also weighted by ω_i.

The learning process of individuals has a deep impact on the GGGP evolution. It shapes the evolutionary process by reinforcing those individuals (derivation trees) which have a propitious genotype to adapt themselves. Those individuals which are far from the optimal solutions will be rapidly replaced. However, those which are close to a solution or which have the adequate genotype to learn the appropriate

characteristics to solve the problem will remain in the population and spread their genome.

The learning process is a short-term adaptation that only takes place during individuals' life. However, ECO enables the incorporation of some learned skills to its individuals genotype so that they can be inherited. There are two possible ways to incorporate new skills to the genotype: directly (Lamarck theory) or indirectly (Baldwin theory). The first method is a theory that offers to the individuals the capacity to directly incorporate learned skills in their genotype. Consequently, the new offspring would not have to learn again the skills its parents have already learned. However, it is possible that those new skills are not so propitious (or even dangerous) and therefore, the offspring is not able to adapt itself as well as if it would not have inherited those new learned skills. The Baldwin theory does not provide any tool for direct codification of learned skills. On the contrary, only those widely spread skills that are beneficial for the individual can be encoded, or partially encoded, in its genotype by means of crossover or mutation operators. This phenomenon overcomes Lamarck theory disadvantage since skills that can be incorporated to the genotype have already been tested because they have been previously learned by a large set of individuals. Moreover, contrary to Lamarck theory, this process is even reversible as parents keep their primitive genome. Therefore, the ECO system seeks benefits from Baldwin Effect instead of Lamarck theory.

In the same way that some animals raise their offspring and take care of them until they mature, the proposed system takes care of new individuals. Every new individual that is incorporated into the GGGP process is protected from environmental pressure. The idea behind this protection is to give each individual an opportunity to adapt itself, by means of the learning process (LS), before it faces the environment. Those individuals are protected until their capacity to learn decreases (Wischmann, et al., 2007), that is, until they reach an adaptation degree according to the capacities provided by their genotype. To measure this capacity to learn, the increase of fitness improvement (Δf) is measured every certain

iterations. If the learning skill is below a given threshold ($\Delta f < t$), the individual is already mature and thus, it is released to the environment. The threshold (t) must be low enough to preserve the life of good solutions and sufficiently high to avoid overprotecting individuals, as this would reduce the performance of the whole system. A setup with threshold $t = \infty$ would behave as a synchronous approach of multilevel optimization. Figure 174 shows the maturity process of two GGGP individuals that have been generated by means of the crossover operator. The offspring remains in a protected state while they try to improve their fitness by means of the learning process. Once they reach the maturity, they are released to the environment.

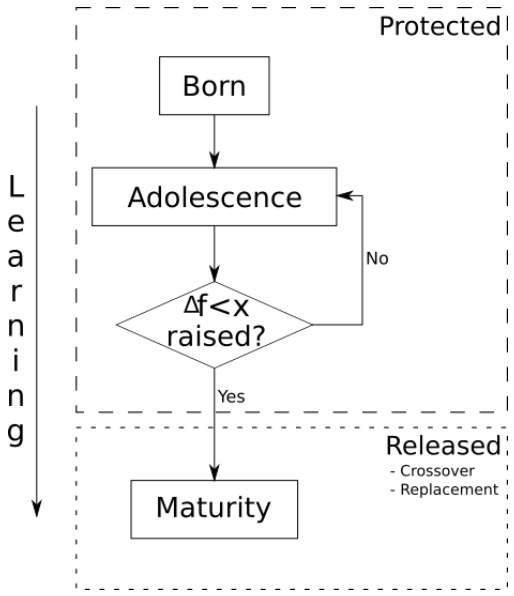

Figure 174: Maturity process of GGGP individuals

7.1. Asynchronous vs Synchronous

Nature typically shows asynchronous-like properties that benefits the global performance of nature processes. Asynchronism helps nature processes keep concurrently working while awaiting for changes that alter the ongoing process. ECO also takes advantage of asynchronous benefits to outperform other multilevel optimization approaches. In order to prove the theoretical asynchronous benefits of ECO, we shall qualitatively compare it to synchronous multilevel optimization

approaches. The following sequence diagram (Figure 175) shows the process differences between ECO (asynchronous) and a synchronous approach to ECO.

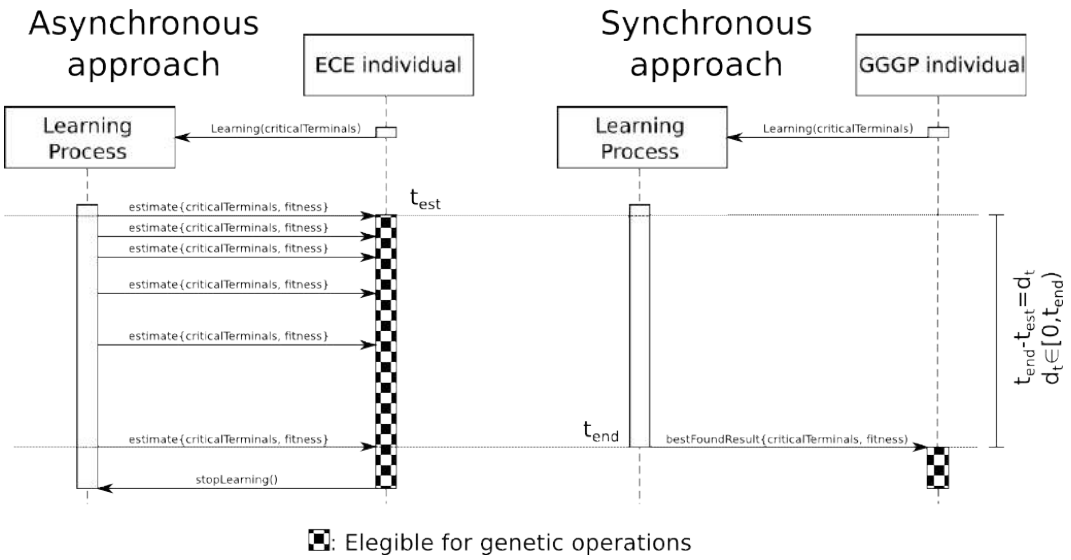

Figure 175: Asynchronous dissertation

On the left of Figure 175 the sequence diagram of a ECO individual is presented. Once an individual is created, its learning process starts. Every time the learning process obtains a new better set of critical terminals, it asynchronously updates the critical terminals values of the derivation tree and the fitness of ECO individual is updated. Since the learning process is continuously and asynchronously providing estimates of individual fitness, ECO evolution is never interrupted. Once a ECO individual obtains its first critical terminals estimate (t_{est}), it is eligible for the GGGP genetic operations (crossover and mutation) once it reaches its mature state. On the right of Figure 175, the sequence diagram of a synchronous approach individual is shown. When a synchronous GGGP individual is created, its learning process starts. However, in this case the evolution process is interrupted for this individual until it is fully optimized. When the learning process ends (t_{end}), the individual fitness and critical terminals are set and from that moment it is eligible for the GGGP genetic operations.

The theoretical difference between ECO and the synchronous approach (d_t), where the learning process is identical for both approaches, is defined as $t_{end} - t_{est}$. t_{end} is the time elapsed since a synchronous GGGP individual is created until it obtains

a fully optimized set of critical terminals. t_{est} represents the time elapsed since a ECO individual is created until it obtains the first estimate of its critical terminals set. During t_{end} and t_{est} time intervals, ECO and synchronous GGGP individuals are halted since derivation trees are uncompleted and their fitness values are not defined. Consequently, they are not part of the evolutionary process and they are not eligible for genetic operations.

Since the learning process is identical for both approaches, it is possible to theoretically define the minimum and maximum values of d_t. The difference d_t is minimized when the first estimate of ECO individuals is obtained when synchronous GGGP approach individuals obtain their critical terminals, that is $t_{end} \approx t_{est}$, then $t_{end} - t_{est} = 0$. On the contrary, the difference is maximized when the first fitness estimate of ECO individuals is obtained close to individuals' creation, $t_{est} \approx 0$, where $t_{end} - t_{est} = t_{end}$. Therefore, we obtain that the theoretical difference d_t is a value that belongs to the interval 0 to t_{end}, $d_t \in [0, t_{end})$. That is, for every new individual the synchronous approach is halted d_t time more than ECO system. Consequently, it is theoretically shown that ECO improves or equals the performance of a synchronous approach provided that learning process is constant for both approaches.

7.2. Partition Based Real-Valued Encoding Scheme for Evolutionary Algorithms

A novel encoding scheme is proposed to always generate vectors (solutions) of dimension l of positive real numbers, which satisfy that each of their components belongs to the interval $[0, k]$, and their sum equals k, $k > 0$ (k restriction): the partition based encoding scheme (PBES) (Font, 2016). These constrains are always guaranteed, independently from the variation operators applied throughout the evolution process. To do so, the evolutionary algorithm works with genotypes in the form $I = (g_1, g_2, \dots g_{l-1})$, $\forall g_i \in [0,1]$, which represents an individual that belongs to the evolutionary algorithm population. PBES provides the following decoding function C:

$$C: [0,1]^{l-1} \rightarrow [0,k]^{l}$$
$$C(I) = I'; \tag{22}$$

where $I = (g_1, g_2, \dots, g_{l-1})$, $\forall g_i \in [0,1]$ and $I' = (g'_1, g'_2, \dots, g'_l)$, $\forall g'_i \in [0,k]$ and $\sum_{i=1}^{l} g'_i = k$, $k > 0$. Therefore, the evolutionary algorithm works with m-sized populations of codified vectors (genotypes) $I_1, I_2, \dots, I_m$of dimension $l-1$ on which genetic operators are applied. The fitness function acts on the decoded version of each I_i, namely I'_i, which is a vector of dimension l that satisfies that the sum of its components is exactly k (k restriction). I'_i is calculated by the decoding function $C(I_i) = I'_i$, so that the fitness function can obtain a measure of how adequate the solution I'_i for the problem at hand is. The Figure 176 shows an example of an individual I_i of dimension 4 ($l = 5$) whose components belong to the interval $[0,1]$. Its corresponding decoded vector I'_i of dimension 5, whose components belong to the interval $[0,5]$ and add up to 5 ($k = 5$), is obtained after applying the decoding function C defined in PBES.

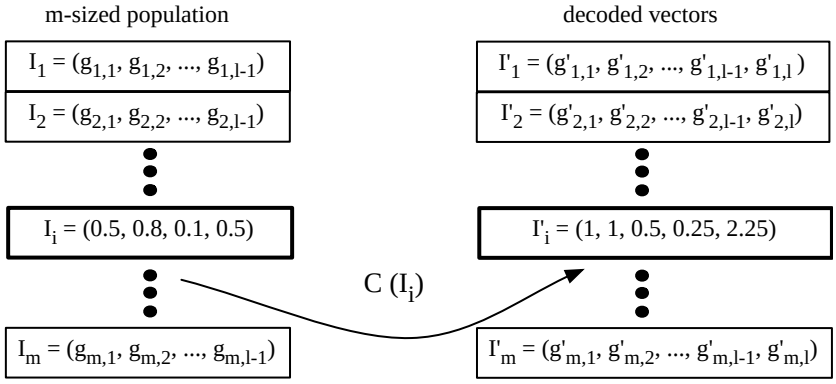

Figure 176: On the left, the m-sized population of individuals where the genetic operators take place (search space). On the right, solutions obtained as a result of applying the decoding function C (solutions space or, in this case, the set of feasible solutions).

Given a positive real valued interval $[0,k] \in \mathbb{R}^+$, where individuals I have to be decoded into I', and an individual I (genotype) as described above, the decoding function C works as follows to obtain I' (the correspondent feasible solution). In order to better illustrate this decoding process, the same example shown in Figure 176 is used, where $I = (0.5, 0.8, 0.1, 0.5) = (g_1, g_2, \dots, g_{l-1})$ and $k = 5$. That is,

solutions are vectors in the form $I' = (g_1', g_2', \dots, g_l')$ of dimension 5 ($l = 5$), whose components belong to the interval $[0,5]$ and add up to 5 (k restriction is 5).

1. The real number line within interval $[0,1]$ is intersected through point g_1, and as a result, it is divided into two segments, $S_1 = [0, g_1]$ and $S_2 = [g_1, 1]$, as Figure 177 shows. For the example given $g_1' = 0.5$.

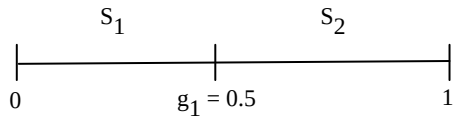

Figure 177: g_1 divides the interval $[0,1]$ into two segments: S_1 and S_2.

2. If g_2 in I exists, the first segment S_1 is divided into two segments: S_{11} y S_{12}. The length of the segment S_{11}, $L(S_{11})$, is calculated as: $L(S_{11}) = g_2 \cdot L(S_1)$. The length of the segment S_{12}, $L(S_{12})$, is the difference between the lengths of S_1 and S_{11}: $L(S_{12}) = L(S_1) - L(S_{11})$. Therefore, as shown in Figure 178, the interval $[0,1]$ is divided into three segments: $S_{11} = [0, g_2 \cdot L(S_1)]$, $S_{12} = [g_2 \cdot L(S_1), g_1'']$ and S_2: $[g_1, 1]$.

   In the example given, $g_2 = 0.8$, therefore $L(S_{11}) = 0.8 \cdot 0.5 = 0.4$, and $L(S_{12}) = 0.5 - 0.4 = 0.1$. Figure 178 shows these results graphically.

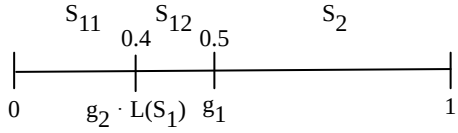

Figure 178: g_2 divides the segment S_1 into two segments: S_{11} y S_{12}.

3. If g_3 in I exists, the segment S_2 is divided into two segments S_{21} y S_{22} as shown in bullet point 2.
4. For each of the remaining g_i of I, the same procedure described in bullet point 2 is applied, consecutively to the segments created in the interval $[0,1]$ from left to right, dividing the interval into l segments.

   In the example given, the interval $[0,1]$ is divided into 5 segments: $S_{111} = [0, 0.2]$, $S_{112} = [0.2, 0.4]$, $S_{12} = [0.4, 0.5]$, $S_{21} = [0.5, 0.55]$ and $S_{22} = [0.55, 1]$.

5. The resulted vector $I'' = (g_1'', g_2'', \dots, g_l'')$ is calculated, which represents each of the lengths of the segments calculated in the previous bullet points. For the given example, $I'' = (0.2, 0.2, 0.1, 0.05, 0.45)$.
6. Vector I'' is unstandardized to obtain $I' = (g_1', g_2', \dots, g_l')$, where $\forall g_i' = g_i'' \cdot k$, which is the result (feasible solution) of applying the decoding function C to vector I (genotype). The decoded vector I' is a candidate solution to the problem (feasible) as it complies with k restriction: $\forall g_i' \in [0, k]$ and $\sum_{i=1}^{l} g_i' = k$. In the case of our example, the vector $I' = (1, 1, 0.5, 0.25, 2.25)$ is obtained, whose components belong to the interval $[0,5]$, and their sum is equal to 5.

Table 12 shows the decoding process pseudocode. This function takes as input a genotype of the population in form $I = (g_1, g_2, \dots, g_{l-1})$and returns a decoded vector $I' = (g_1', g_2', \dots, g_l')$, which represents a feasible solution to the problem at hand. The while loop splits the interval $[0,1]$ into l subintervals, as described in steps 1 to 4. Then, the list must be sorted according to the lower limit of each subinterval. The following for loop (step 5) obtains the vector $I'' = (g_1'', g_2'', \dots, g_l'')$ by calculating the length of each l subinterval. Finally, the last *for* loop (step 6) calculates the feasible solution I' to the problem at hand from I''.

Table 12: Pseudocode of the decoding process described in bullet points 1 to 6.

Input: $k, I = (g_1, g_2, \dots, g_{l-1}), \forall g_i \in [0,1]$
Output: $I' = (g_1', g_2', \dots, g_l'), \forall g_i' \in [0, k]$ *and* $\sum_{i=1}^{l} g_i' = k$

// Implementation of the steps 1 to 4 of the decoding process
list = {[0,1]}
i = 1 // index for I
while i <= l-1 do {
e = remove (1, list)// first element of list is assigned to e and removed from the list
split e into
S = [lowerLimit (e), lowerLimit (e) + gi· length (e)],
S' = [upperLimit (S), upperLimit (e)]
appendElement (S, list) // add S to the end of the list
appendElement (S', list) // add S' to the end of the list

```
        i = i + 1
    }
    sort (list) // in ascending order according to lower limits
```

```
    // Implementation of the step 5 of the decoding process
    Vector I'' = () // the vector of lengths of the segments in the list
    for i = 1 to l do {
        gi'' = length (element (i, list))
        add gi'' to I''
    }
```

```
    // Implementation of the step 6 of the decoding process
    Vector I' = (); // the output vector
    for i = 1 to l do {
        gi' = gi''·k
        add gi' to I'
    }
    return I'
```

It is important to underline at this point that the evolutionary algorithm works with populations of individuals (genotypes) in the form $I = (g_1, g_2, \dots g_{l-1})$, $\forall g_i \in [0,1]$. The decoding function C is only applied once for each new generated individual throughout the evolutionary process to calculate its fitness. Therefore, there is no need to design or use new specific genetic operators, especially for those that generate new individuals: initialization process, crossover or mutation. They only must satisfy that all the individuals being generated are real-number vectors of dimension $l - 1$, whose alleles belong to the interval $[0,1]$. When individuals are generated under these conditions, which are simple, it is guaranteed that they are feasible individuals: the decoding function C will generate their corresponding solutions that comply with the constraints for the problem at hand.

Another important feature of the proposing encoding scheme is its fairness. Neither the values within the genotype I, nor the decoded solution I', are statistically bigger or smaller than the others. That is, there are no biases, e.g. because of their position in the vectors. Genotypes are randomly and uniformly generated (on initialization process), randomly modified (by mutation operator) or recombined (by crossover

operator). No biases are expected, except for those caused by these genetic operators. The decoded version of a genotype is not biased either since the decoding process is proportional. Codifications within the space solution I' usually insert strong biases, since components of individuals are forced to comply with k restriction.

7.3. Results

A set of experiments has been conducted to show the goodness of the proposed methods ECO and PBES. PBES is firstly applied to train Naïve Bayes Classifiers that aim to solve different experiments from the UCI repository (Lichman, 2013). A covariance matrix adaptation evolution strategy (CMA-ES) (Hansen, 2006) is employed to determine the values of the conditional probability tables. PBES is compared with to a direct encoding scheme (DES), so that, the individuals of the CMA-ES contain vectors encoding the conditional probability tables in the case of PBES, or vectors with the probabilities of the conditional probability tables in the case of DES. In the first case, every individual contains a feasible solution. In the second case, a repair operator must be applied to ensure the probabilities of an attribute add up 1. The fitness function is the misclassification rate $[0,1]$. The evolutionary process seeks to minimize the fitness value. The evolutionary process converges if it every instance of the dataset is successfully classified or if it accomplishes 50 generations without improving the average population fitness.

Secondly, optimization experiments have been conducted to compare the performance of ECO and a synchronous approach of ECO optimization. Both uses a CMA-ES with PBES as the local search optimization algorithm. Different experiments are accomplished to show the performance of the both approaches when trying to solve different UCI problems with a Naïve Bayes Classifier. A GGGP optimization process is employed to define the dependencies between the attributes and the class of the Naïve Bayes Classifier. To such aim, the CFG G_{NB} is designed (23). This CFG encodes a small search space since it is not recursive. The

axiom directly generates each of the dependencies of the classifier (r_1). Then, a binary value determines whether the dependency exists.

$$
\begin{aligned}
&G_{NB} = (V, \Sigma, R, N)\\
&V = \{B\}\\
&\Sigma = \{1\ 0\}\\
&R = \{\\
&\qquad r_1 \quad S ::= B \cdots B\\
&\qquad r_2 \quad B ::= 0\\
&\qquad r_3 \quad B ::= 1\\
&\}
\end{aligned}
\tag{23}
$$

No additional features are included to the misclassification rate fitness function of the CMA-ES. The evolutionary process seeks to minimize the fitness value. The evolutionary processes converge if every instance of the dataset is successfully classified or if they accomplish 50 generations without improving the average population fitness. The maturity threshold t is set to 0.2. GUPI and EDX are respectively employed as population initialization and variation operator. The EDX chooses 50% of the population to produce an offspring whose size is 25% of the whole population size. The learning rate is also set to 0.3 and the smoothing rate to 0.001. The tournament selection is applied to improve the population diversity. No mutation or immigration are applied. Population size is fixed to 100. 50 runs are performed for each experiment. Comparisons have been applied in terms of best fitness of the population, number of iterations, and number of learning iterations. A statistical study of these values is conduced as it is defined in section 5.2.2.

7.3.1. Partition based encoding scheme experiments

The performed experiments show PBES achieves an important improvement in terms of iterations for most of the tested problems. In particular, PBES outperforms for iris problem (Figure 180), cancer problem (Figure 182), diabetes problem (Figure 184), hypo-thyroid problem (Figure 186), credit-a problem (Figure 188), anneal problem (Figure 190), and digits problem (Figure 192). Statistical evidences are provided to state there is a statistically significant difference between the iterations means ($p < 0.05$). However, the PBES only improves the average

best fitness for three of the seven problems: the credit-a problem (Figure 187), anneal problem (Figure 189), and digits problem (Figure 191). Contrary to previous results, the PBES shows a significantly higher average best fitness value for the diabetes problem (Figure 183). In the rest of the cases, the iris problem (Figure 179), cancer problem (Figure 181), and hypo-thyroid problem (Figure 185), no statistical evidences are provided to state there is a statistically significant difference between the iterations means ($p < 0.05$).

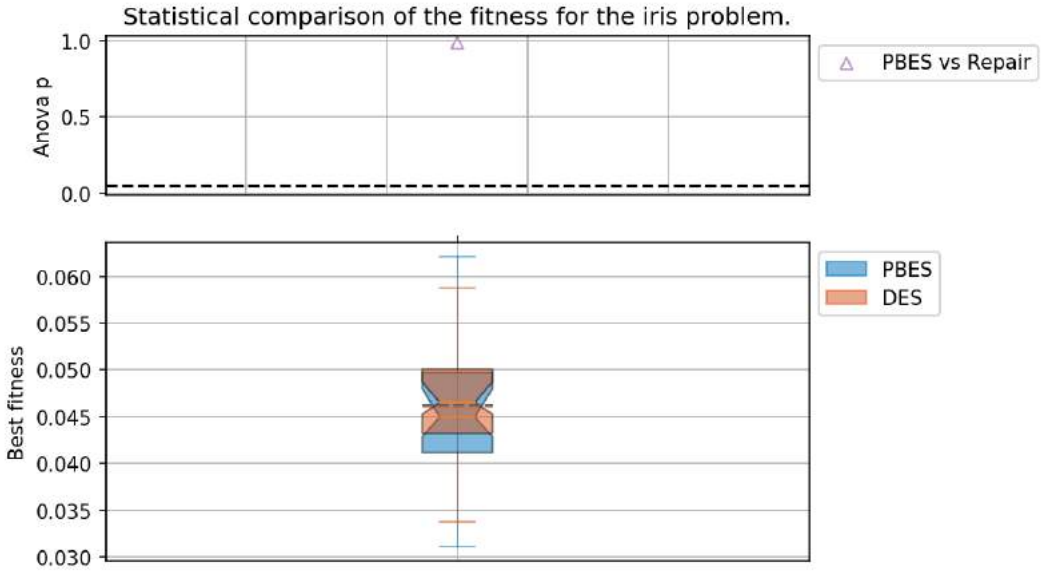

Figure 179: One-way ANOVA of the average fitness to compare PBES with DES when using Naïve Bayes classifiers to solve iris problem. 50 executions run per target derivation tree.

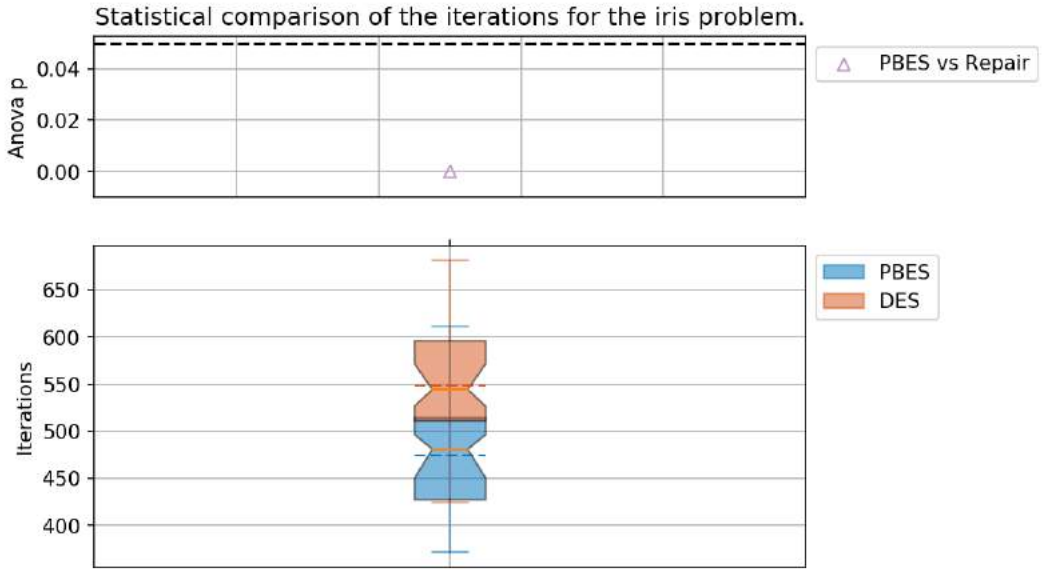

Figure 180: One-way ANOVA of the average iterations to compare PBES with DES when using Naïve Bayes classifiers to solve iris problem. 50 executions run per target derivation tree.

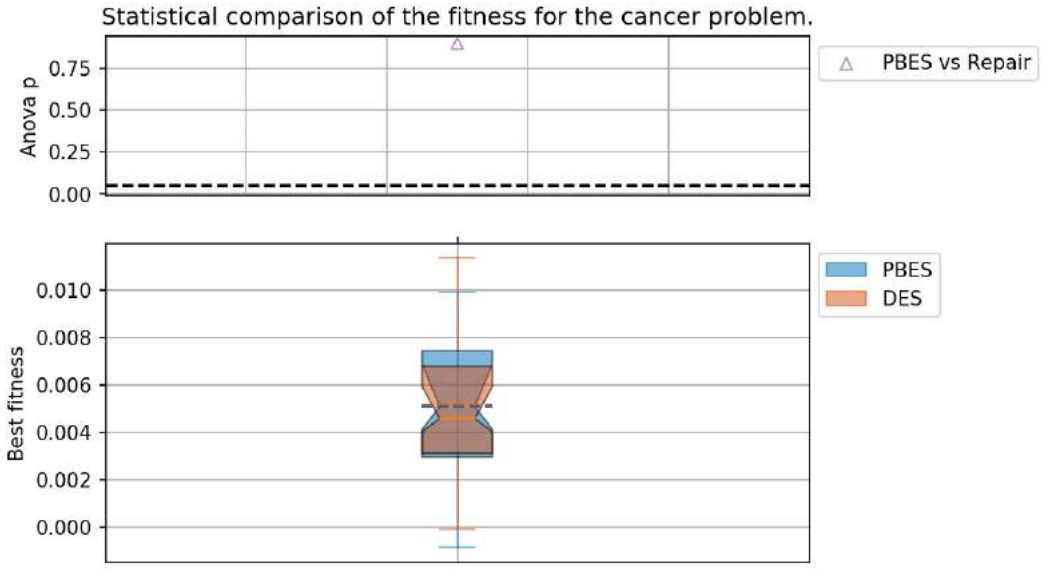

Figure 181: One-way ANOVA of the average fitness to compare PBES with DES when using Naïve Bayes classifiers to solve cancer problem. 50 executions run per target derivation tree.

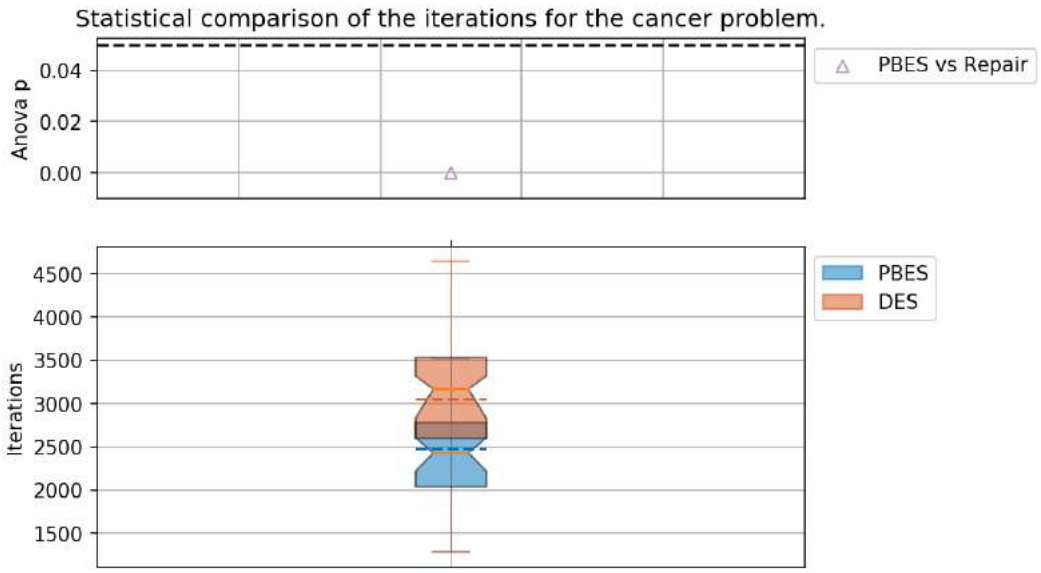

Figure 182: One-way ANOVA of the average iterations to compare PBES with DES when using Naïve Bayes classifiers to solve cancer problem. 50 executions run per target derivation tree.

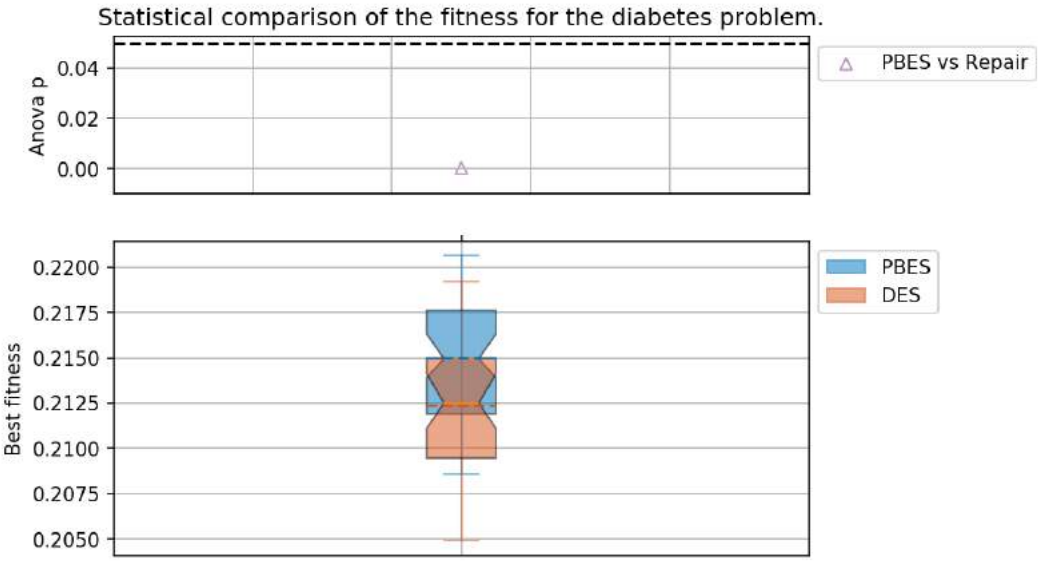

Figure 183: One-way ANOVA of the average fitness to compare PBES with DES when using Naïve Bayes classifiers to solve diabetes problem. 50 executions run per target derivation tree.

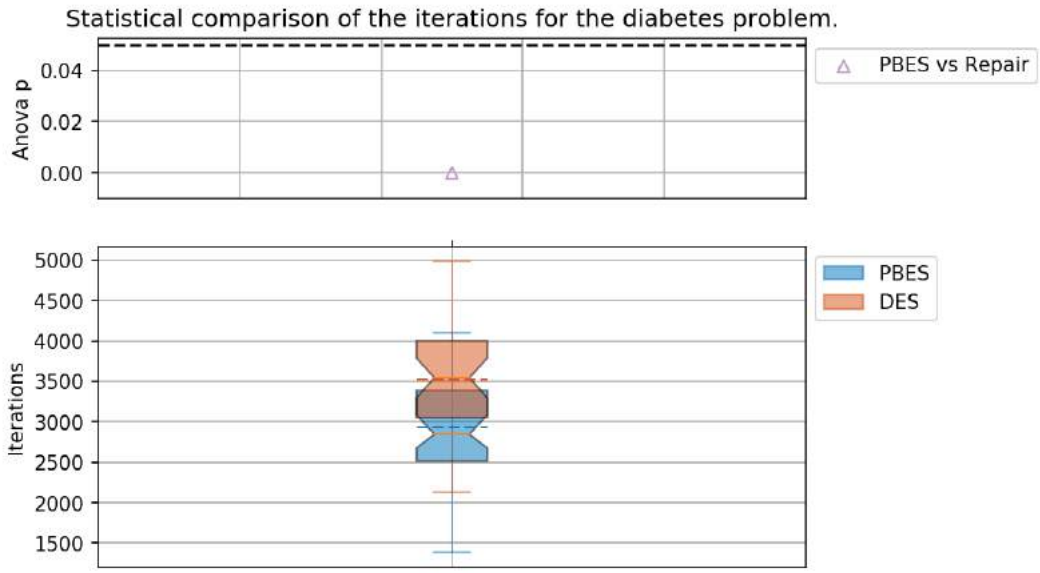

Figure 184: One-way ANOVA of the average iterations to compare PBES with DES when using Naïve Bayes classifiers to solve diabetes problem. 50 executions run per target derivation tree.

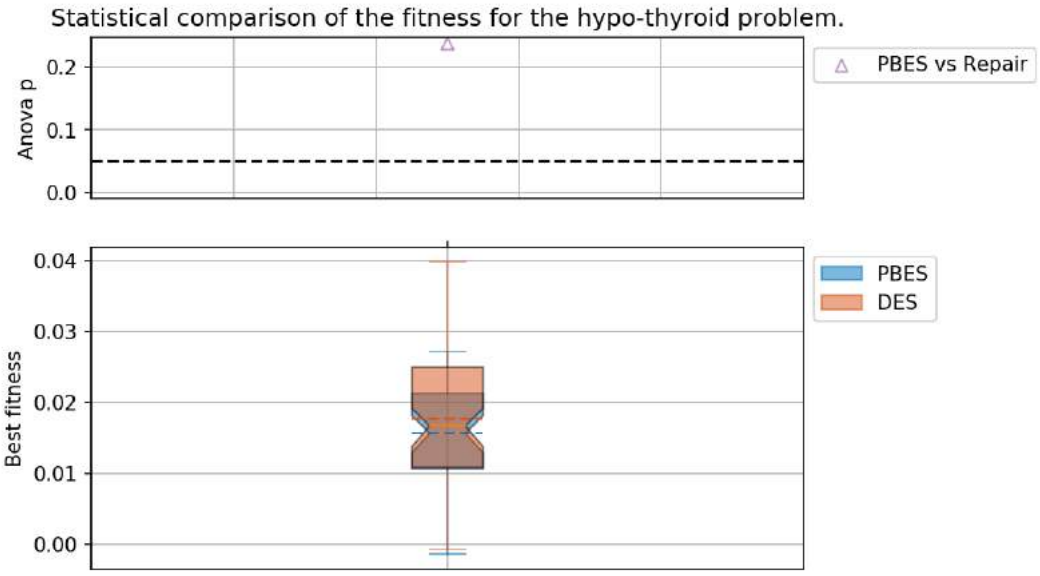

Figure 185: One-way ANOVA of the average fitness to compare PBES with DES when using Naïve Bayes classifiers to solve hypo-thyroid problem. 50 executions run per target derivation tree.

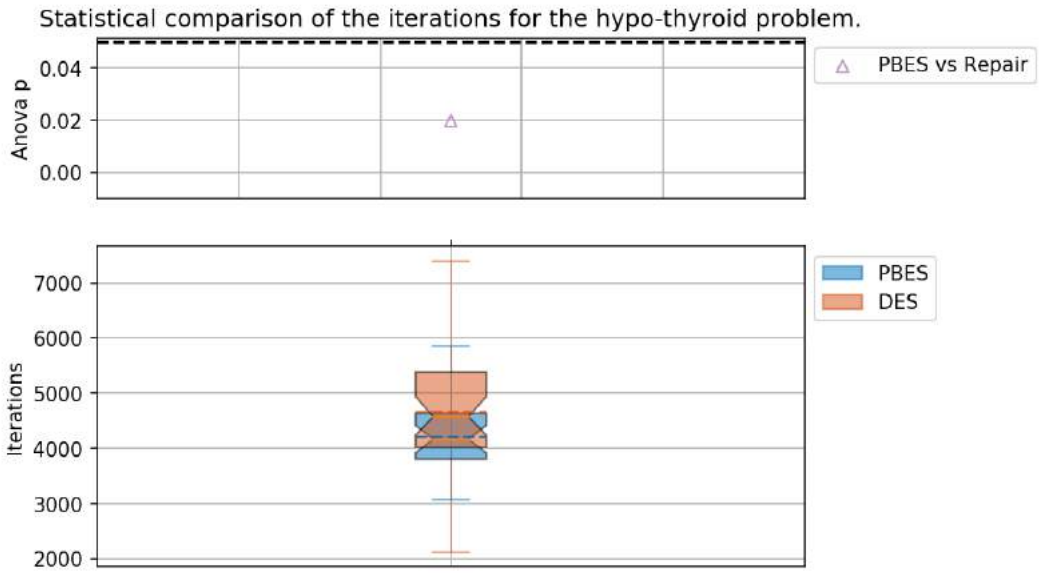

Figure 186: One-way ANOVA of the average iterations to compare PBES with DES when using Naïve Bayes classifiers to solve hypo-thyroid problem. 50 executions run per target derivation tree.

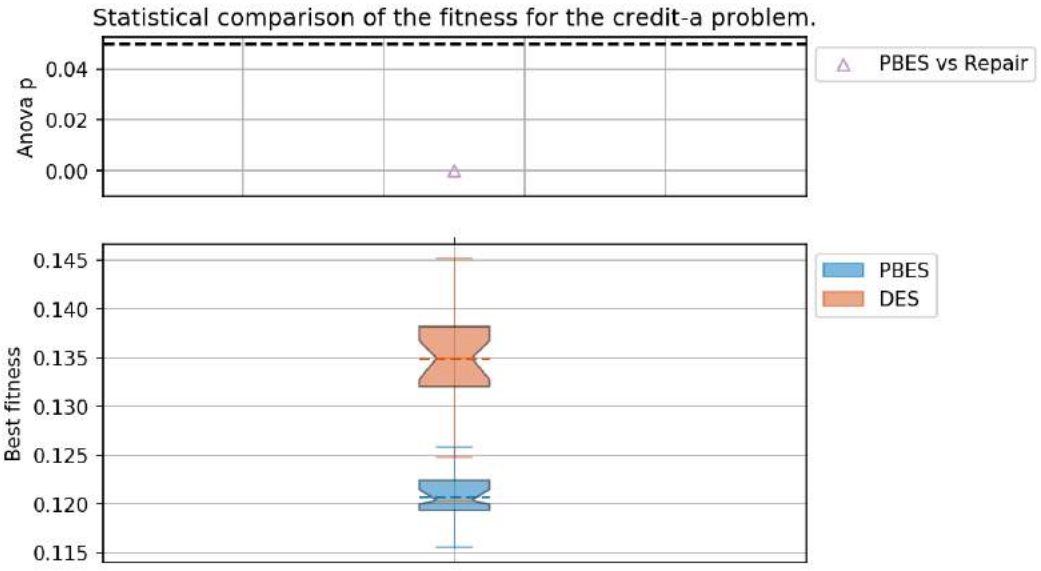

Figure 187: One-way ANOVA of the average fitness to compare PBES with DES when using Naïve Bayes classifiers to solve credit-a problem. 50 executions run per target derivation tree.

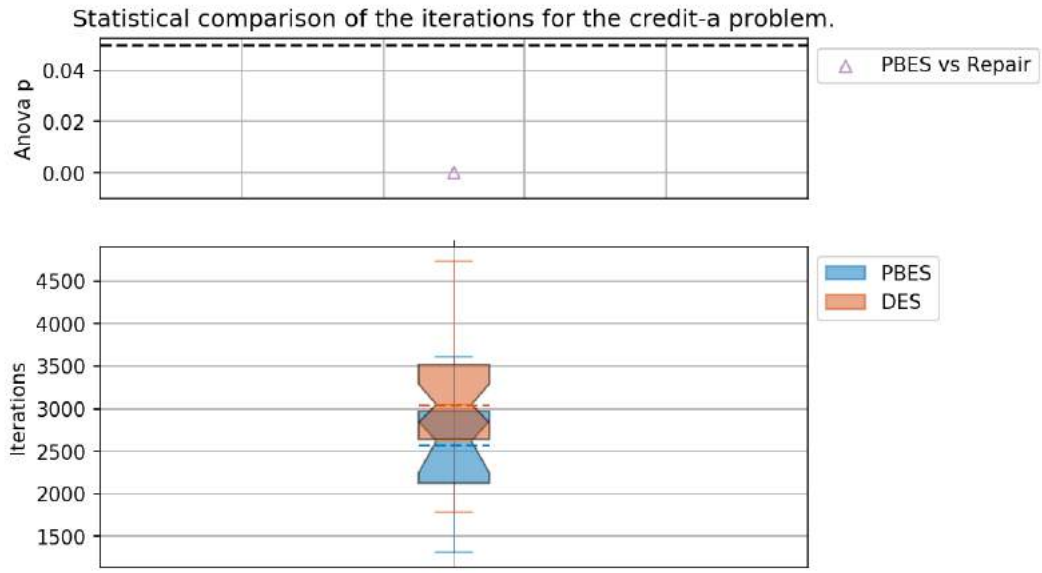

Figure 188: One-way ANOVA of the average iterations to compare PBES with DES when using Naïve Bayes classifiers to solve credit-a problem. 50 executions run per target derivation tree.

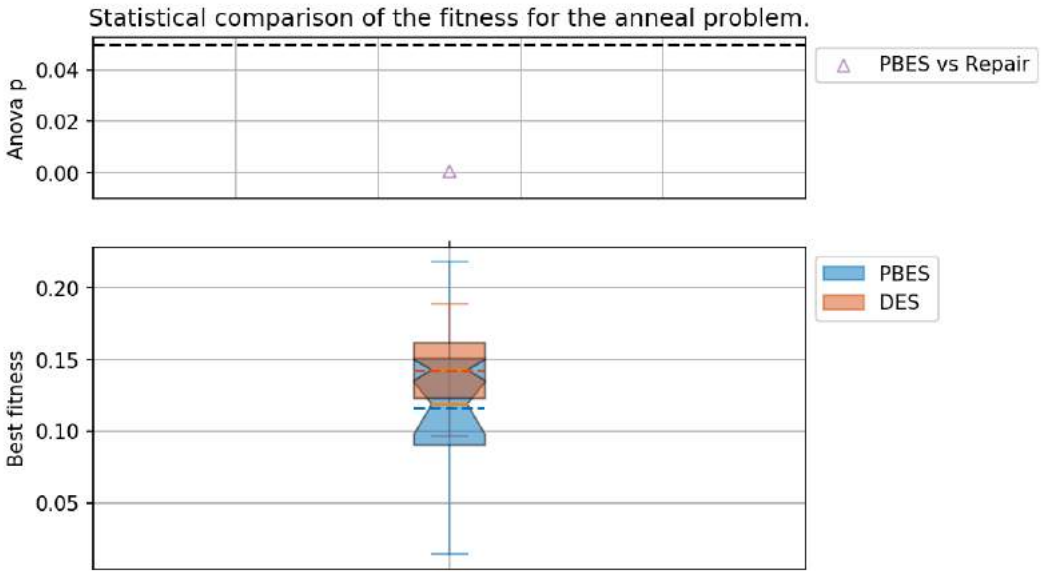

Figure 189: One-way ANOVA of the average fitness to compare PBES with DES when using Naïve Bayes classifiers to solve anneal problem. 50 executions run per target derivation tree.

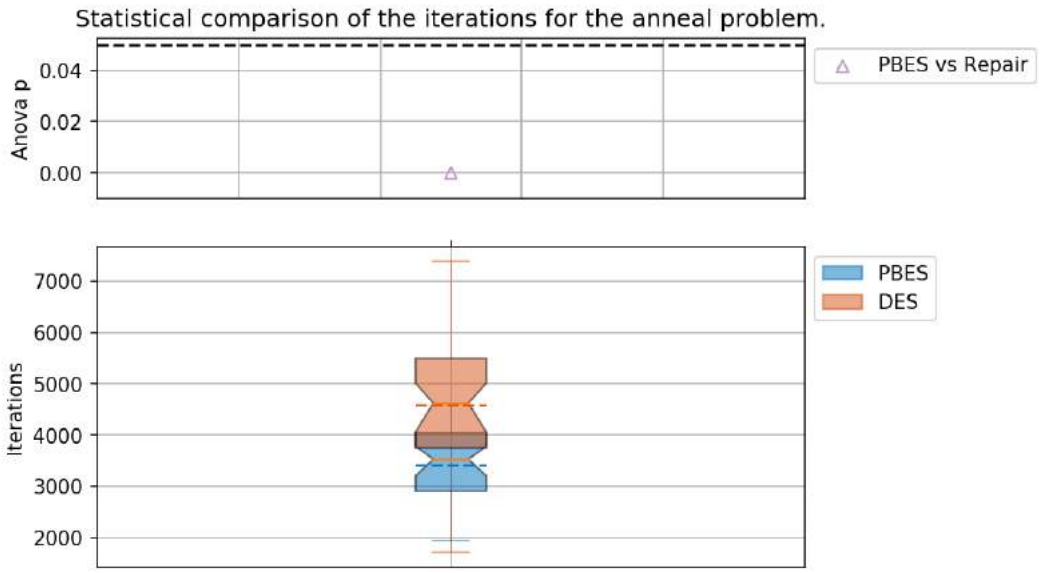

Figure 190: One-way ANOVA of the average iterations to compare PBES with DES when using Naïve Bayes classifiers to solve anneal problem. 50 executions run per target derivation tree.

Figure 191: One-way ANOVA of the average fitness to compare PBES with DES when using Naïve Bayes classifiers to solve digits problem. 50 executions run per target derivation tree.

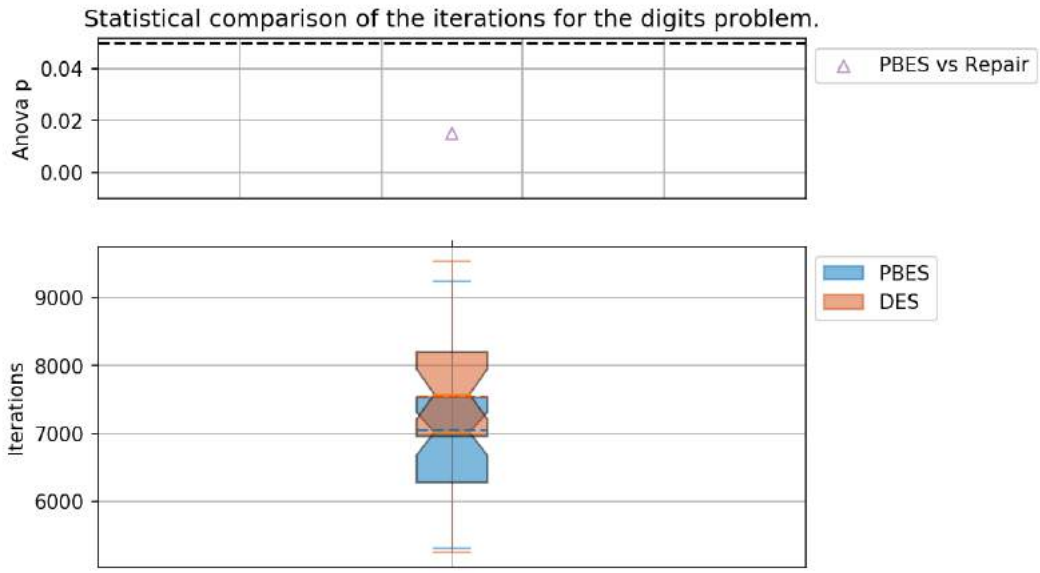

Figure 192: One-way ANOVA of the average iterations to compare PBES with DES when using Naïve Bayes classifiers to solve digits problem. 50 executions run per target derivation tree.

7.3.2. Endosymbiotic co-optimization experiments

The results of the performed experiments comparing an ECO approach with a synchronous ECO approach, shows that, in general terms, the ECO approach maintains the average best fitness and GGGP iterations while it reduces the learning iterations. It is shown that there is slightly rise in the fitness value and the GGGP iterations. However, no statistical evidences are provided to state there is a statistically significant difference between the fitness means or the GGGP iterations means ($p < 0.05$) in most of the cases. In particular ECO maintains the fitness and GGGP iterations levels for the iris problem (Figure 193 and Figure 194), cancer problem (Figure 196 and Figure 197), diabetes problem (Figure 199 and Figure 200), hypo-thyroid problem (Figure 202 and Figure 203) and anneal problem (Figure 208 and Figure 209) while it reduces the learning iterations (Figure 195, Figure 198, Figure 201, Figure 204, and Figure 210 respectively). However, for credit-a and digits problems the best fitness is proven to be significantly higher (Figure 205 and Figure 211 respectively). However ECO outperforms the synchronous approach in terms of number of iterations in the credit-a problem (Figure 206 and Figure 207). In the case of digits problem, there is no significant difference (Figure 212 and Figure 213).

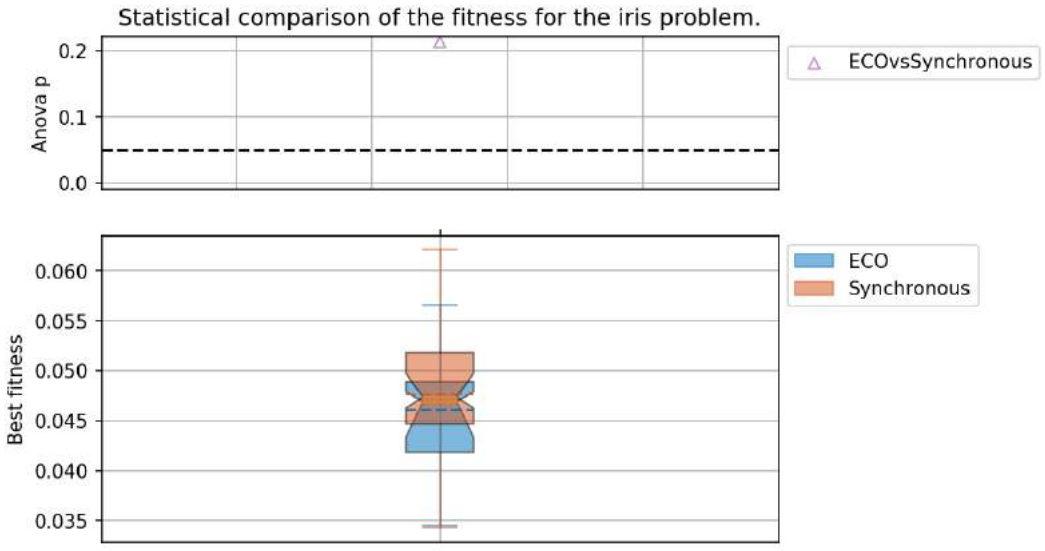

Figure 193: One-way ANOVA of the average fitness to compare ECO and a synchronous approach when using G_{NB} to solve iris problem. 50 executions run per target derivation tree.

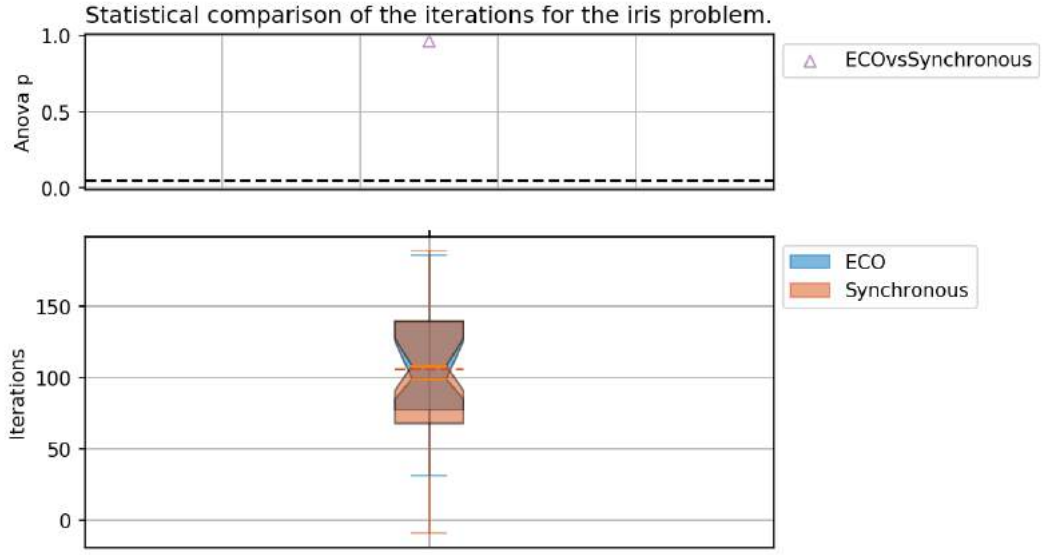

Figure 194: One-way ANOVA of the average iterations to compare ECO and a synchronous approach when using G_{NB} to solve iris problem. 50 executions run per target derivation tree.

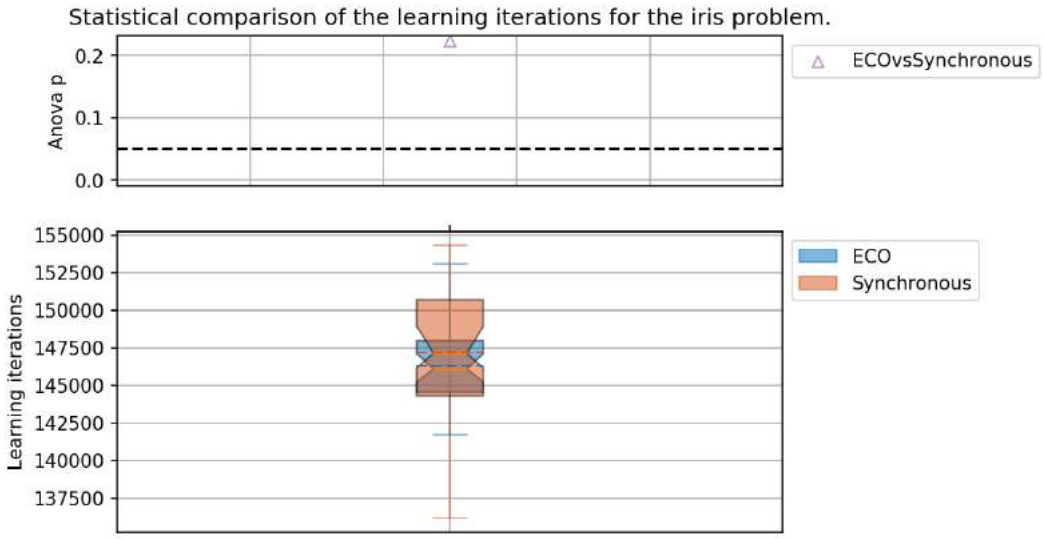

Figure 195: One-way ANOVA of the average learning iterations to compare ECO and a synchronous approach when using G_{NB} to solve iris problem. 50 executions run per target derivation tree.

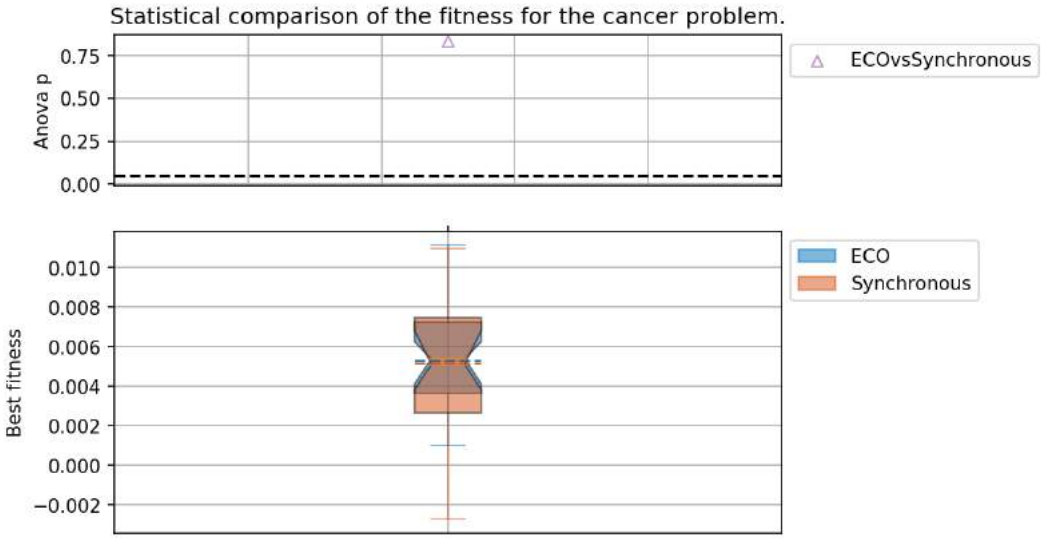

Figure 196: One-way ANOVA of the average fitness to compare ECO and a synchronous approach when using G_{NB} to solve cancer problem. 50 executions run per target derivation tree.

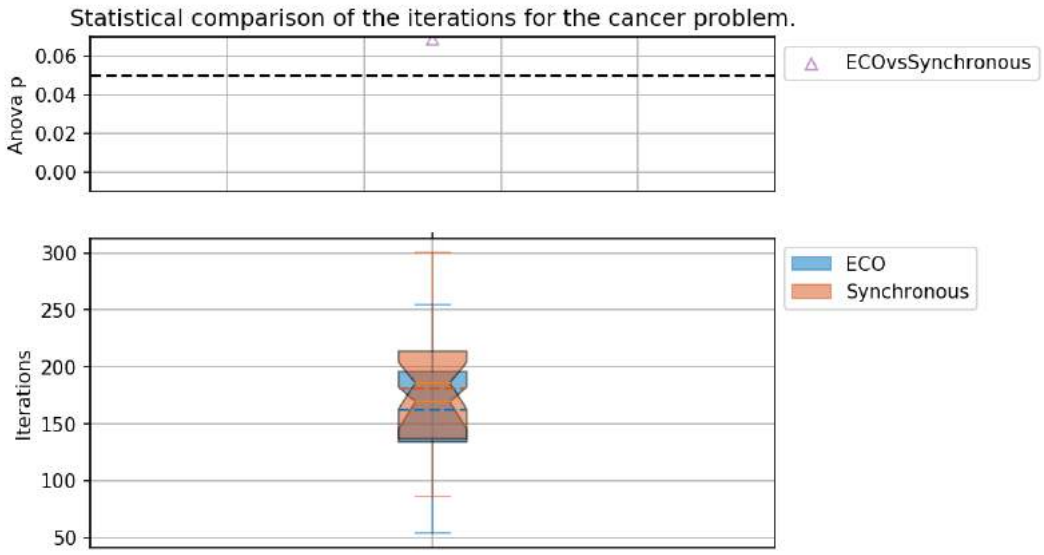

Figure 197: One-way ANOVA of the average iterations to compare ECO and a synchronous approach when using G_{NB} to solve cancer problem. 50 executions run per target derivation tree.

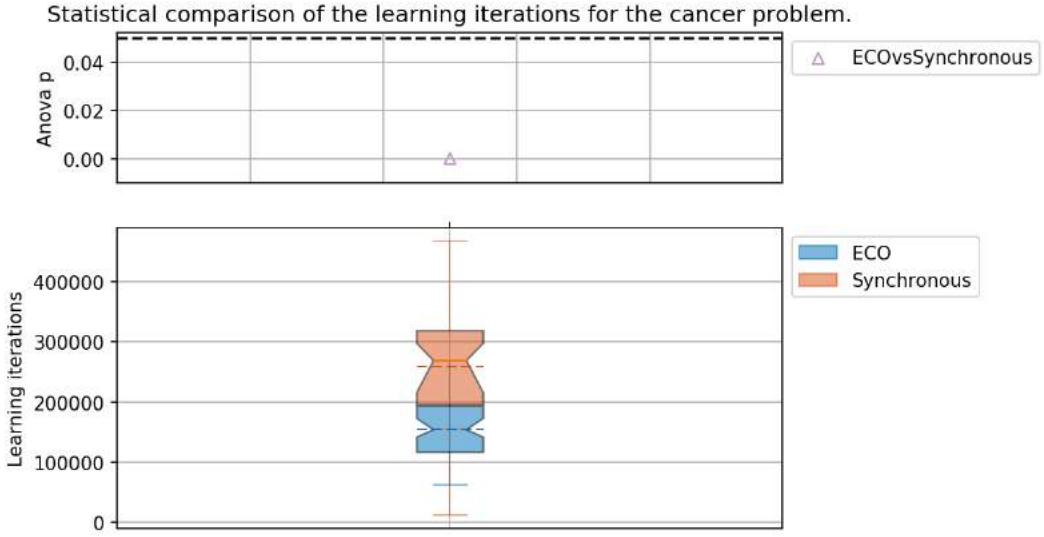

Figure 198: One-way ANOVA of the average learning iterations to compare ECO and a synchronous approach when using G_{NB} to solve cancer problem. 50 executions run per target derivation tree.

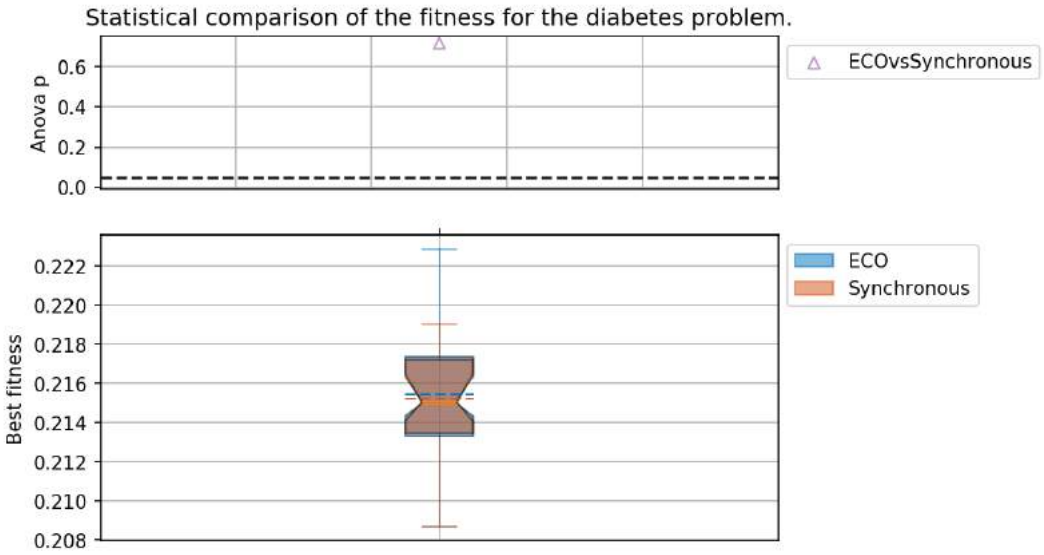

Figure 199: One-way ANOVA of the average fitness to compare ECO and a synchronous approach when using G_{NB} to solve diabetes problem. 50 executions run per target derivation tree.

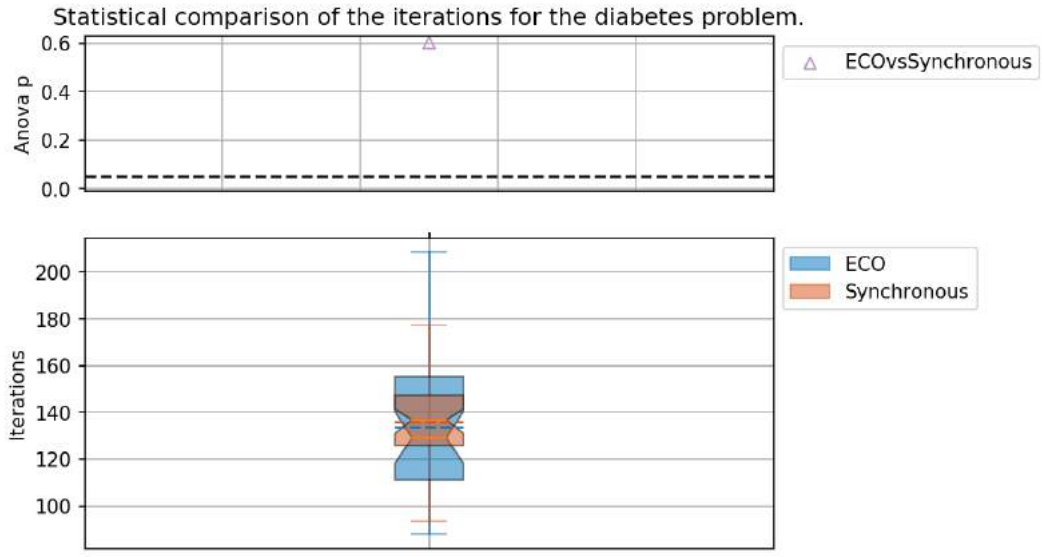

Figure 200: One-way ANOVA of the average iterations to compare ECO and a synchronous approach when using G_{NB} to solve diabetes problem. 50 executions run per target derivation tree.

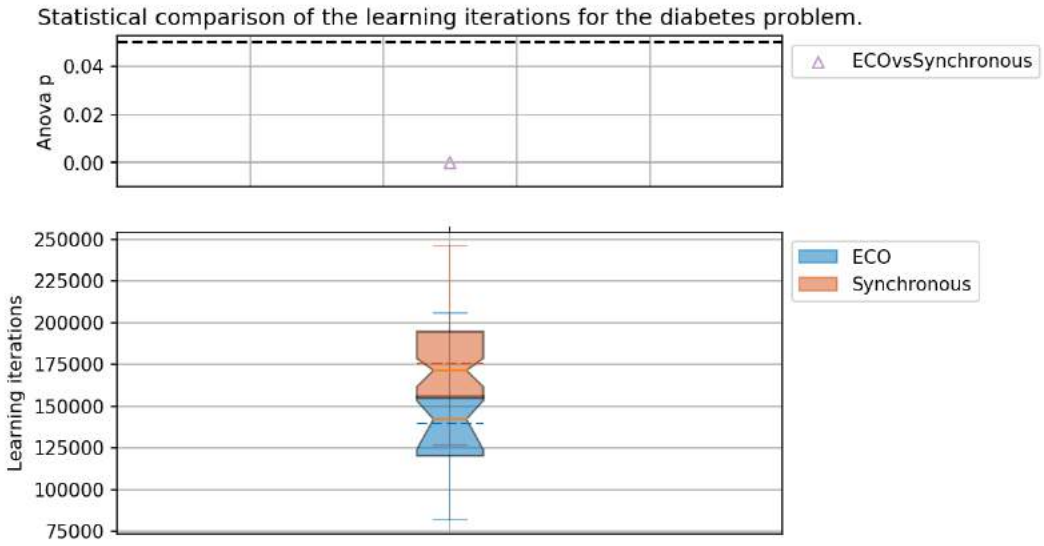

Figure 201: One-way ANOVA of the average learning iterations to compare ECO and a synchronous approach when using G_{NB} to solve diabetes problem. 50 executions run per target derivation tree.

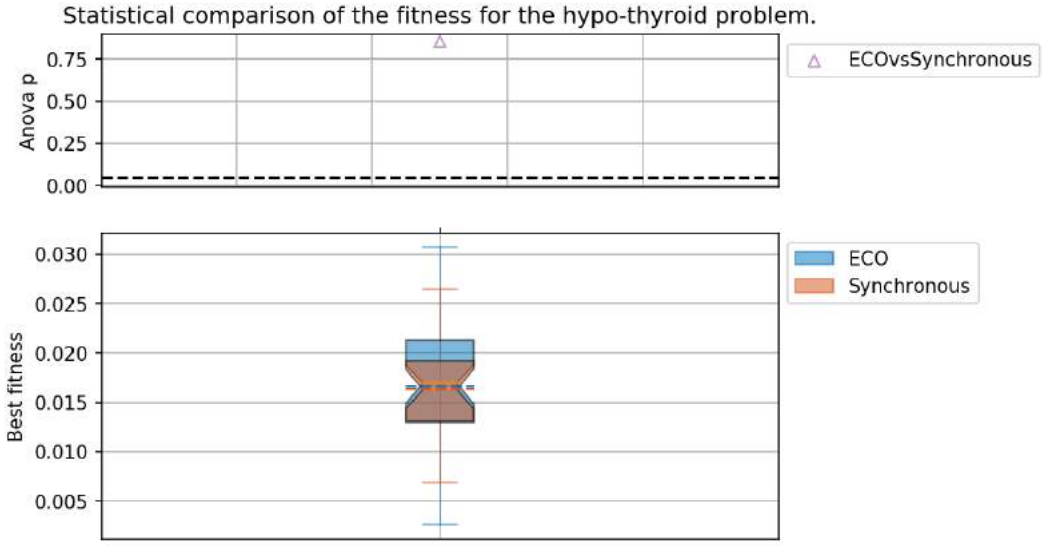

Figure 202: One-way ANOVA of the average fitness to compare ECO and a synchronous approach when using G_{NB} to solve hypo-thyroid problem. 50 executions run per target derivation tree.

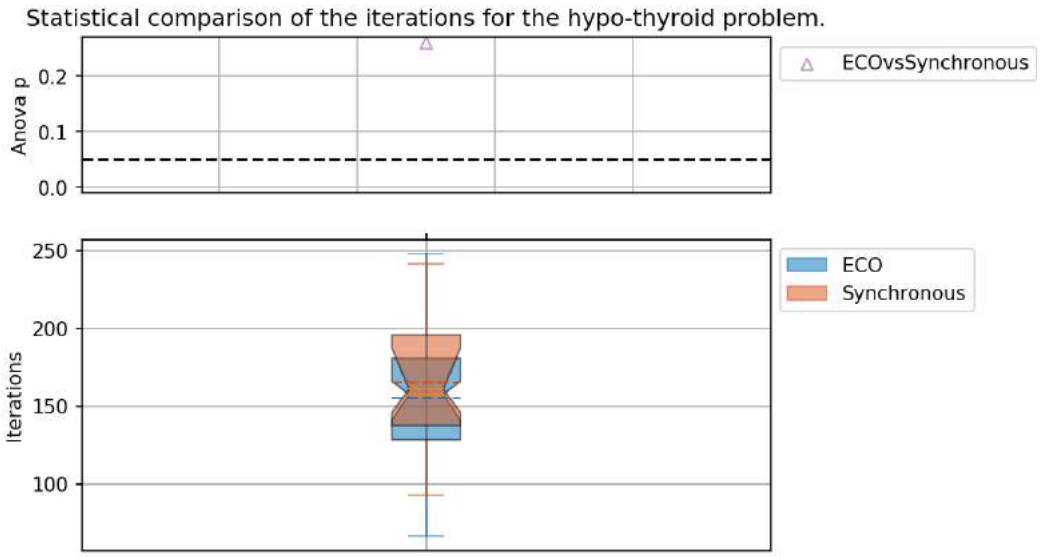

Figure 203: One-way ANOVA of the average iterations to compare ECO and a synchronous approach when using G_{NB} to solve hypo-thyroid problem. 50 executions run per target derivation tree.

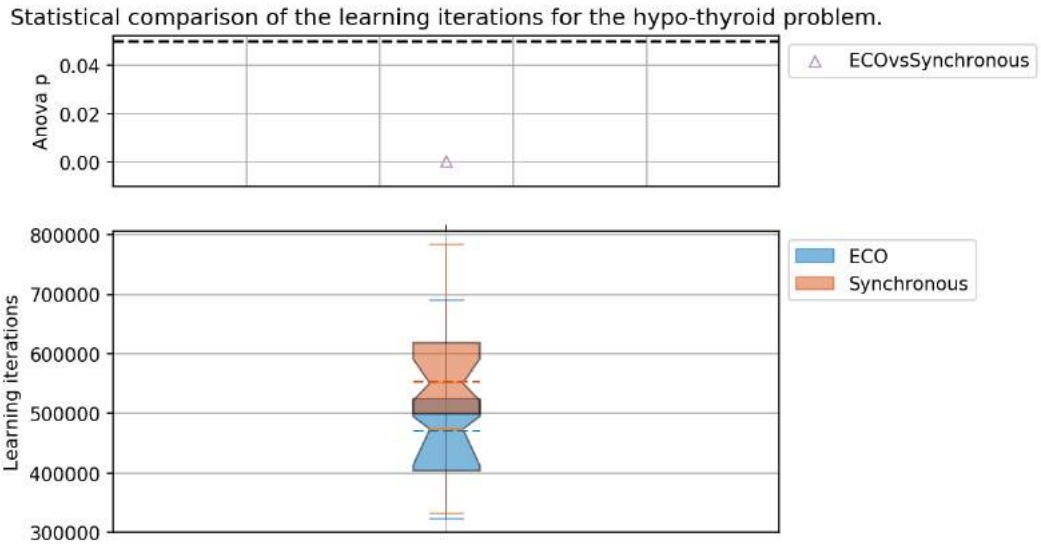

Figure 204: One-way ANOVA of the average learning iterations to compare ECO and a synchronous approach when using G_{NB} to solve hypo-thyroid problem. 50 executions run per target derivation tree.

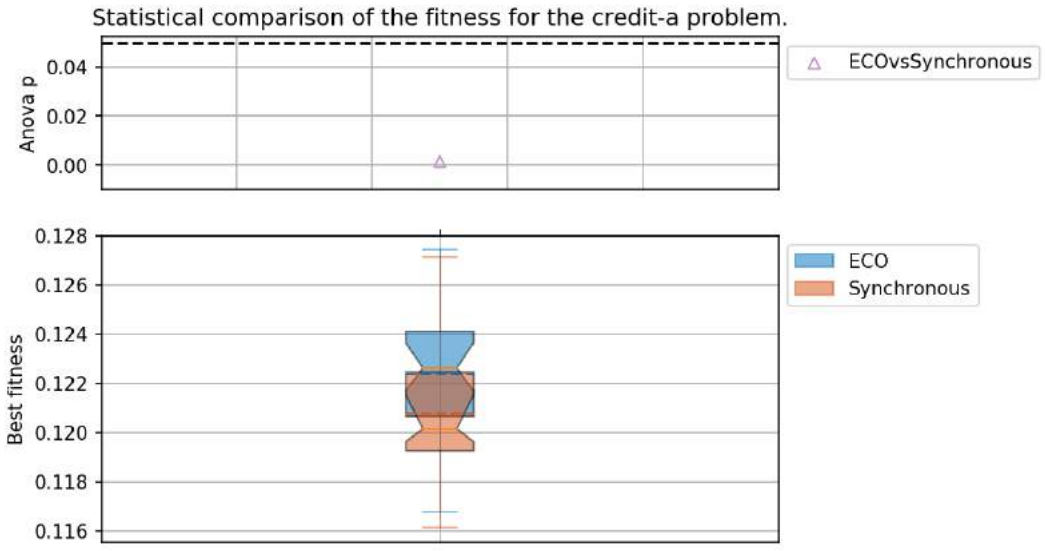

Figure 205: One-way ANOVA of the average fitness to compare ECO and a synchronous approach when using G_{NB} to solve credit-a problem. 50 executions run per target derivation tree.

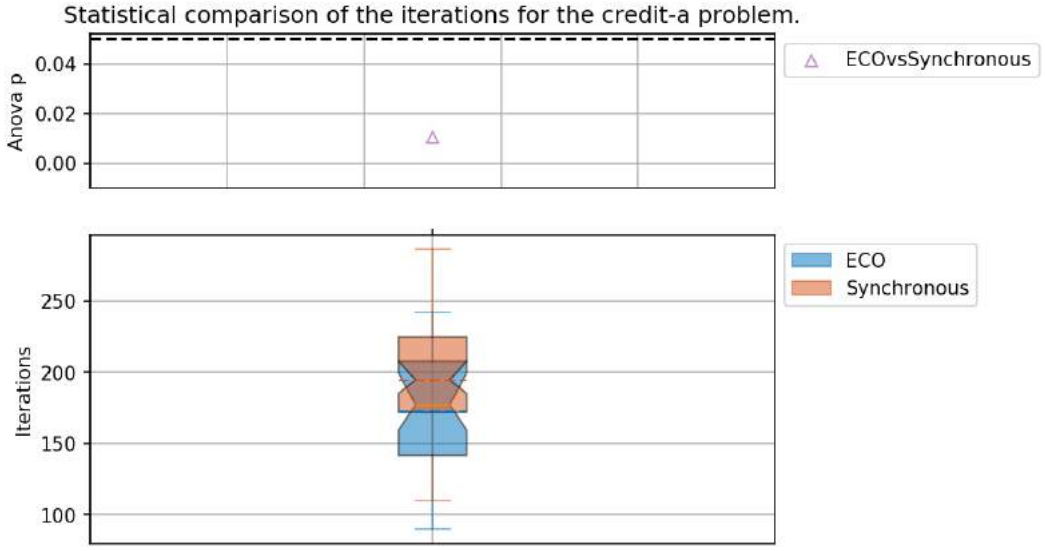

Figure 206: One-way ANOVA of the average iterations to compare ECO and a synchronous approach when using G_{NB} to solve credit-a problem. 50 executions run per target derivation tree.

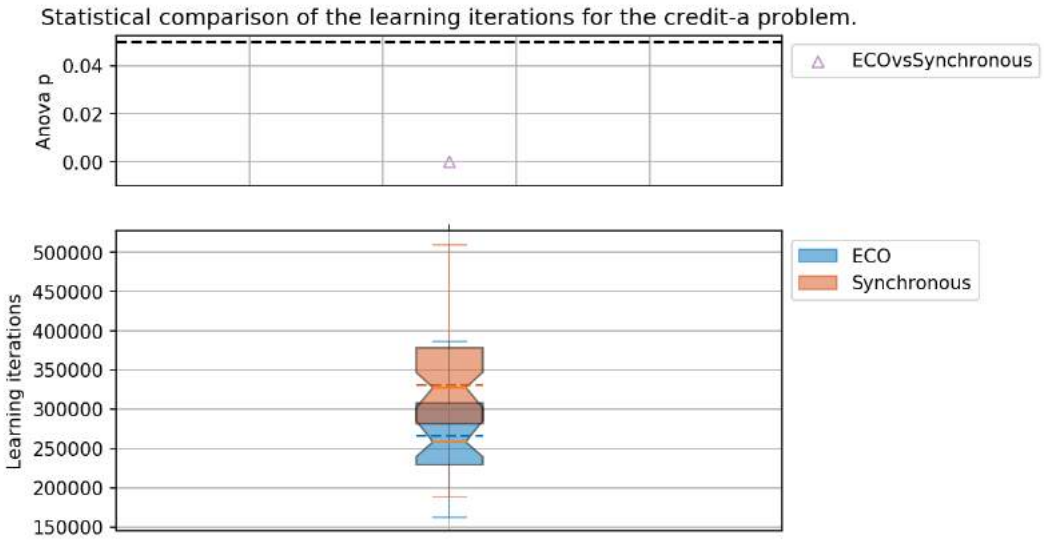

Figure 207: One-way ANOVA of the average learning iterations to compare ECO and a synchronous approach when using G_{NB} to solve credit-a problem. 50 executions run per target derivation tree.

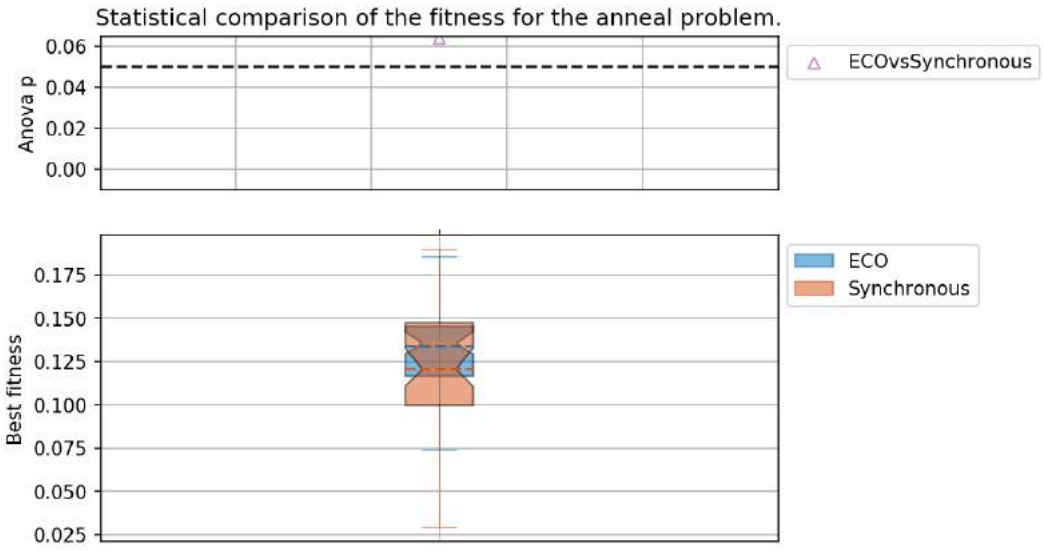

Figure 208: One-way ANOVA of the average fitness to compare ECO and a synchronous approach when using G_{NB} to solve anneal problem. 50 executions run per target derivation tree.

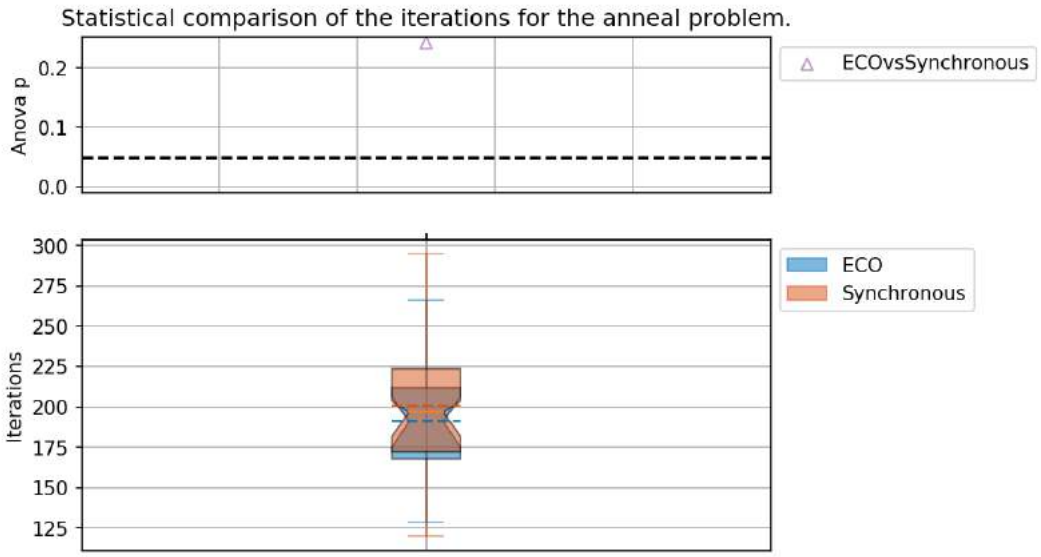

Figure 209: One-way ANOVA of the average iterations to compare ECO and a synchronous approach when using G_{NB} to solve anneal problem. 50 executions run per target derivation tree.

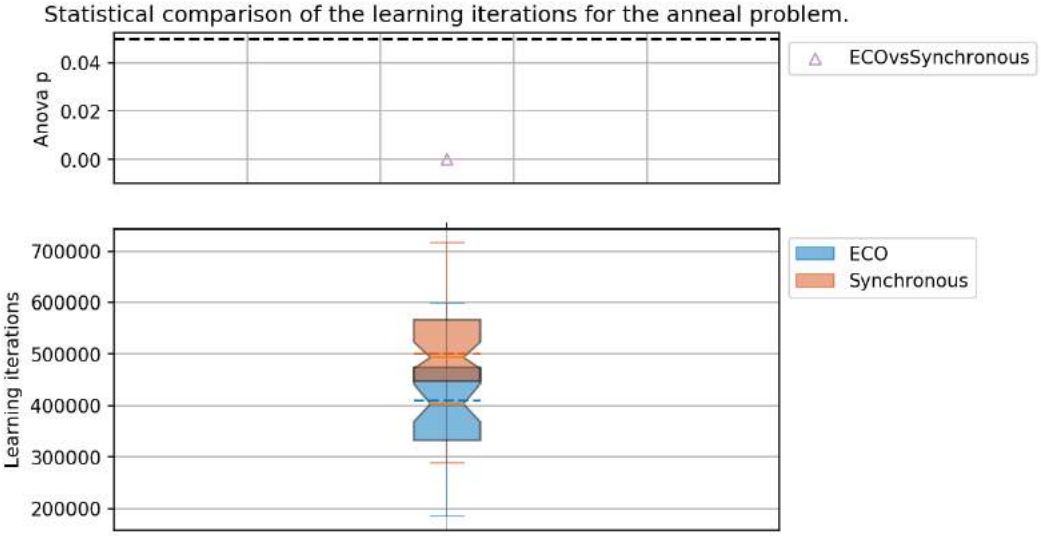

Figure 210: One-way ANOVA of the average learning iterations to compare ECO and a synchronous approach when using G_{NB} to solve anneal problem. 50 executions run per target derivation tree.

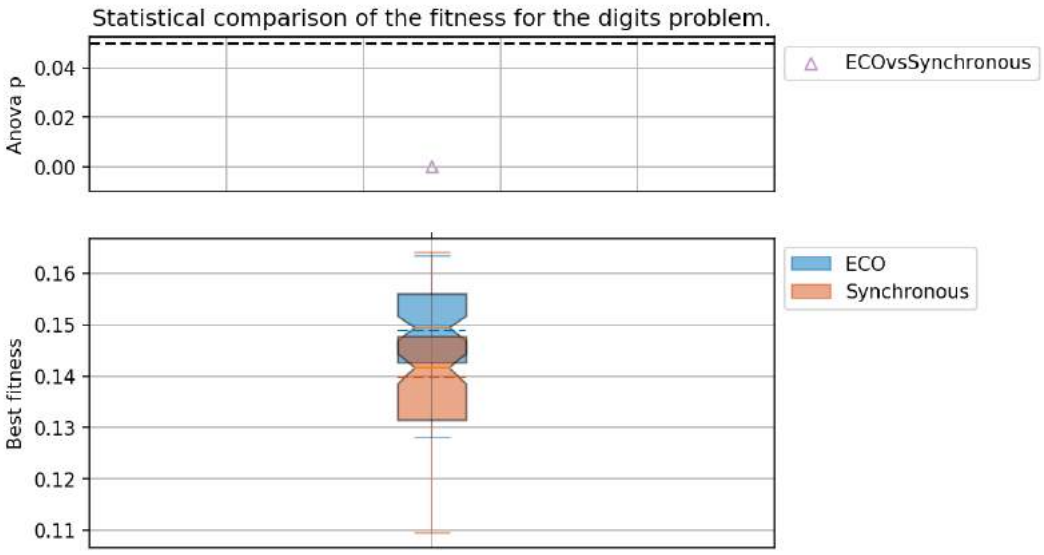

Figure 211: One-way ANOVA of the average fitness to compare ECO and a synchronous approach when using G_{NB} to solve digits problem. 50 executions run per target derivation tree.

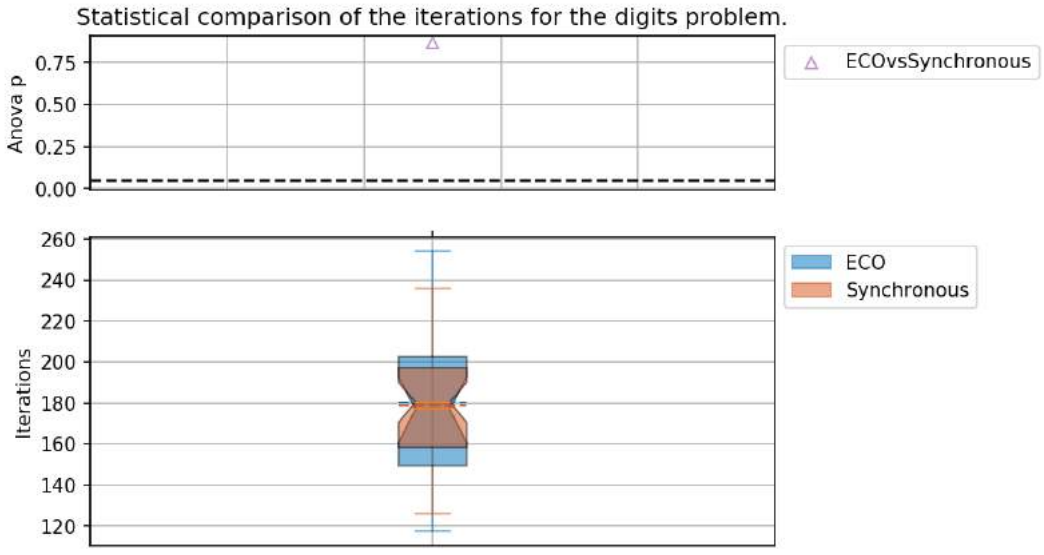

Figure 212: One-way ANOVA of the average iterations to compare ECO and a synchronous approach when using G_{NB} to solve digits problem. 50 executions run per target derivation tree.

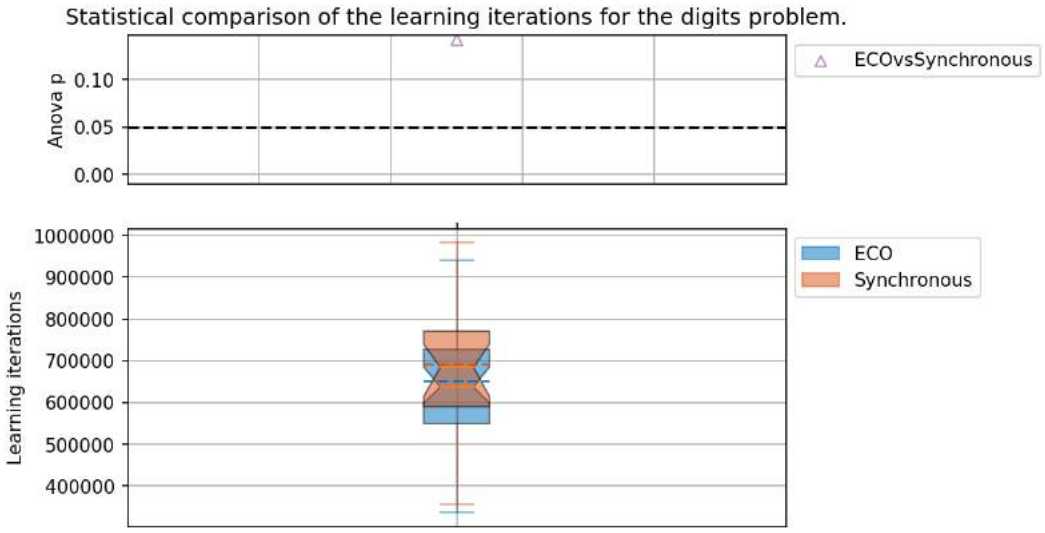

Figure 213: One-way ANOVA of the average learning iterations to compare ECO and a synchronous approach when using G_{NB} to solve digits problem. 50 executions run per target derivation tree.

V. CONCLUSIONS AND FUTURE RESEARCH

8. Conclusions

The research work presented in this thesis has been focused on the study of the limitations of GGGP when solving problems encoded by CFG whose search space is large. In particular, it has been detected performance limitations when the CFG produces large derivation trees or it has a high cardinality terminal symbol set. According to the research performed, the population initialization and the variation operators are the main causes of these limitations. The previously existing population initialization methods bias the initialization towards individuals belonging to some small areas of the search space, what may lead to generate no representative samples of the search space. In general, this causes the premature convergence to undesirable suboptimal solutions. Two different variation approaches were studied: Whigham crossover and a EDA approach. The WX shows a very explorative behaviour. Although exploratory operators provide a mechanism to reduce the probability to converge to local optima, an excessive exploration may drastically reduce the performance of the evolutionary process or even randomize the optimization process reducing the probability of finding optimal solutions. The EDA approach shows a destructive behaviour that reduce the diversity of population. This diversity reduction causes in general premature convergence to undesirable suboptimal solutions. These limitations are aggravated when the size of the derivation trees generated by the CFG is large or the cardinality of the terminal symbol set is high.

To overcome these limitations, three novel approaches are presented: a grammatically uniform population initialization (GUPI), a population estimation of distribution crossover operator (EDX) and an asynchronous endosymbiotic co-optimization technique (ECO and PBES). The GUPI is an initialization method designed to produce representative samples of the search space encoded in the CFG. To such aim, the population initialization analyses the CFG to generate individuals that are first uniformly distributed in terms of number of recursive derivations and then in the search space defined by the CFG. Since initial populations generated

by GUPI are more representative of the CFG search space, there is a performance improvement in the overall evolutionary process associated to the reduction of the exploratory cost. The EDX operator is variation operator inspired in EDA optimization mechanism that generate new population individuals containing the most representative characteristics of a subset of selected promising individuals of the population. Contrary to EDAs generational replacement, in the presented crossover operator only a small subset of individuals is produced and replaced in each evolutionary iteration. This setup reduces the diversity loss and consequently it avoids the premature convergence to suboptimal solutions. The ECO method provides an asynchronous mechanism for parallel optimization of the derivation tree structure and the terminal symbols that drastically reduces the computational cost of optimization methods when CFGs contains terminal symbol sets with a high cardinality. In addition to the inherent computational time reduction associated to the parallel computation of the optimization, the presented method presents a computational cost reduction associated to the estimation of the individuals' fitness. Instead of halting the optimization process until the terminal symbols are optimized, the estimations provide a mechanism to guide the evolutionary process while individuals are optimized. Since individuals are replaced during the evolutionary process, there is a computational cost reduction associated to the pending terminals optimization of the replaced individuals that synchronous approaches do not provide. Together with ECO, a novel encoding scheme called PBES is presented to avoid the generation of infeasible individuals when terminals represent the components of a real numbered vector and these components are required to sum a predefined constant.

The performed experiments have shown the advantages of using the proposed approaches when dealing with problems with the abovementioned characteristics. A set of laboratory and real world problems experiments have been performed to test the efficacy of each of these methods. CFGs encoding some intelligent systems have been employed to test the performance in problems with large search spaces. In relation to GUPI, the experiments have shown an important performance

improvement associated to the production of more representative samples of the population. Performance improvements have been also achieved with EDX compared to WX and EDA. The EDX shows a fast convergence compared to WX avoiding the diversity loss of EDA. ECO has shown a computational cost reduction when dealing with high cardinality terminal symbol sets. Moreover, it is remarkable the fact that even for terminal symbol sets with a barely high cardinality, the ECO is already effective. The PBES has also shown performance improvements that are strictly related to the avoidance of infeasible individual penalization or elimination. In general terms, it has been shown an improvement of the presented approaches compared to some of the common current GGGP approaches. These improvements are more notorious when dealing with large search spaces like those related to deep learning methods.

8.1. Contributions and Hypothesis Corroboration.

This subsection provides a structured description of the contributions made by the research work presented in this thesis. The main contributions are enumerated below according to their type.

Theoretical Contributions and Hypothesis corroboration

- The GGGP performance is reduced when evolving large derivation trees or the cardinality of the set of terminal symbols is high.
- The GGGP limitations when dealing with CFG whose search space is large, concretely, when the CFG generates large derivation trees and the cardinality of the terminal symbol set is high, are caused by population initialization and variation operators.
- Current GGGP population initialization is biased according to CFG definition.
- The Whigham crossover does not adequately guide the evolutionary process due to its highly explorative behavior.
- The generational replacement of EDAs approaches for GGGP can cause diversity loss and increment the computational cost.

- The generation of representative sample in GGGP population initialization improves the optimization process performance.
- A crossover based on population estimation of distribution with an enhanced exploitative behavior that avoids the EDA's generational replacement adequately guide the evolutionary process and improve the performance.
- The application of local search optimization techniques, when the cardinality of terminal symbol sets is high, improves the overall optimization process performance.
- The fitness estimation is sufficient to determine individuals' goodness and guide the evolutionary process.
- The application of custom encoding schemes that do not generate infeasible individuals improves the overall optimization process performance.

Solutions proposed

- A new population initialization that generates more representative samples of the population: grammatical uniform population initialization method (GUPI).
- A new crossover operator that adequately guides the evolutionary process to promising solutions and keeps the population diversity: the estimation of distribution crossover (EDX).
- A new cooperative optimization method for the terminal symbols that provides estimation of individuals' fitness to reduce the computational cost: the endosymbiotic co-optimization method (ECO).
- A new encoding scheme for real numbered vectors whose components must sum a predefined constant: the partition-based encoding scheme (PBES).

9. Future Research

As result of the research work performed, some promising research lines have arisen. Firstly, the elaboration of new mutation operators for EDX that improve the exploratory search. Current mutation operators are not designed to explore the search space areas spatially located between population individuals. The successful attainment of this research line would produce significance improvements of the performance of GGGP evolutionary process when dealing with complex search spaces. However, the definition of methods that explore such areas is deem to be complicated. As a starting point, the smoothing method of EDX could be modified to actively change the smoothing rate according to the evolutionary inertia. Concretely, the average population fitness can be used to determine the variations of the smoothing rate.

Taking advantage of the characteristics of the grammar search space definition presented for the estimation of distribution crossover, the grammar expansion, new optimization methods could be applied. In particular, ant colony optimization emerges as a promising optimization technique that have been successfully applied to other EDAs approaches. Thus, it is expected to obtain interesting results in this matter.

Following with grammar expansion characteristics, a new research line may arise from the modification of this search space to encode and optimize graphs. Grammatically uninform population initialization and estimation of distribution crossover methods could be adapted to work with a graph encoding scheme and perform an evolutionary optimization process. This approach may improve the performance of evolutionary processes when trying to solve problems where graphs are involved such as artificial neural networks, decision trees, or Bayesian networks. Moreover, the adaptation of the presented methods to the graph paradigm may facilitate the application of ant-colony optimization algorithms or other graph-oriented methods.

VI. BIBLIOGRAPHY

Bibliography

Abbass H. A. Hoai N. & Mckay R. I. (2002). AntTAG: A new method to compose computer programs using colonies of ants. *In The IEEE Congress on Evolutionary Computation,* Citeseer, pp. 1654--1659.

Abbass H. A. Hoai X. & Mckay R. I. (2002). AntTAG: A new method to compose computer programs using colonies of ants. *In Proceedings of the 2002 Congress on Evolutionary Computation, 2002.,* IEEE, pp. 1654-1659.

Ackley D. & Littman M. (1991). Interactions between learning and evolution. *Artificial life II,* Volume 10, pp. 487-509.

Alonso C. L. Montaña J. L. & Borges C. E. (2009). Evolution strategies for constants optimization in genetic programming. *In Proceedings of ICTAI'09. 21st International Conference on Tools with Artificial Intelligence, 2009.,* IEEE, pp. 703-707.

Alonso L. & Schott R. (2013). *Random generation of trees: random generators in computer science.* Springer Science & Business Media.

Bäck T. Fogel D. B. & Michalewicz Z. (1997). *Handbook of evolutionary computation.* New York:

Bäck T. Fogel D. B. & Michalewicz Z. (1997). *Handbook of evolutionary computation.* Oxford(New York): Taylor & Francis.

Baldwin J. M. (1896). A new factor in evolution. *The american naturalist,* 30(354), pp. 441-451.

Baluja S. (1994). *Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning,*

Barrios Rolanía D. Ríos Carrión J. & Segovia Pérez F. J. (1992). Generalización del operador de cruce en algoritmos genéticos. *In XVIII Conferencia Latinoamericana de Informática,* pp. 137-144.

Bary H. A. (1879). *Die Erscheinung der Symbiose: Vortrag.* Verlag von Karl J. Trübner.

Beaser E. Schwartz J. K. Bell III C. B. & Solomon E. I. (2011). Hybrid Genetic Algorithm with an Adaptive Penalty Function for Fitting Multimodal Experimental Data. *Journal of chemical information and modeling,* 51(9), pp. 2164-2173.

Böhm W. & Geyer-Schulz A. (1996). Exact Uniform Initialization For Genetic Programming. *In Foundations of Genetic Algorithms IV,* University of San Diego, CA, USA, R. K. Belew and M. Vose, p. 379–407.

Booth T. L. & Thompson R. A. (1973). Applying probability measures to abstract languages. *IEEE transactions on Computers,* 100(5), pp. 442--450.

Boryczka M. a. C. Z. J. (2002). Solving Approximation Problems By Ant Colony Programming. *In GECCO Late Breaking Papers,* pp. 39-46.

Bosman P. A. & De Jong E. D. (2004). Grammar transformations in an EDA for genetic programming. *In Proceedings of GECCO 2004 Workshop,* Seattle, WA, USA, pp. 26-30.

Bottou L. (2010). Large-scale machine learning with stochastic gradient descent. *In Proceedings of COMPSTAT'2010,* Springer, pp. 177--186.

Burke E. K. Gustafson S. & Kendall G. (2004). Diversity in genetic programming: An analysis of measures and correlation with fitness. *Evolutionary Computation, IEEE Transactions,* 8(1), pp. 47-62.

Burkhardt R. W. (2013). Lamarck, Evolution, and the Inheritance of Acquired Characters. *Genetics,* 194(4), pp. 793--805.

Castelli M. Vanneschi L. & Popovič A. (2016). Controlling individuals growth in semantic genetic programming through elitist replacement. *Computational intelligence and neuroscience,* Volume 2016, p. 42.

Castelli M. Vanneschi L. & Silva S. (2014). Semantic search-based genetic programming and the effect of intron deletion. *IEEE transactions on cybernetics,* 44(1), pp. 103-113.

Chellapilla K. (1997). Evolving computer programs without subtree crossover. *In IEEE Transactions on Evolutionary Computation,* pp. 209-216.

Cerny B. M. Nelson P. C. & Zhou C. (2008). Using differential evolution for symbolic regression and numerical constant creation. *In Proceedings of the 10th annual conference on Genetic and evolutionary computation,* 2008, ACM, pp. 1195-1202.

Cagnoni S. Rivero D. & Vanneschi L. (2005). A purely evolutionary memetic algorithm as a first step towards symbiotic coevolution. *In The 2005 IEEE Congress on Evolutionary Computation, 2005.,* IEEE, pp. 1156-1163.

Chomsky N. (1956). Three models for the description of language. *Information Theory, IRE Transactions on 2.3,* pp. 113-124.

Chomsky N. (1959). On certain formal properties of grammars. *Information and control 2.2,* pp. 137-167.

Chomsky N. & Schützenberger M. P. (1963). The algebraic theory of context-free languages. *Studies in Logic and the Foundations of Mathematics,* Volume 35, pp. 118--161.

Choubey N. S. (2010). A novel encoding scheme for traveling tournament problem using genetic algorithm. *IJCA Special Issue on Evolutionary Computation,* 2(7), pp. 79-82.

Cobb H. G. & Grefenstette J. J. (1993). *Genetic algorithms for tracking changing environments,*

Coello C. A. (2016). Constraint-handling techniques used with evolutionary algorithms. *In Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion,* ACM, pp. 563-587.

Couchet J. Manrique D. & Porras L. (2007). Grammar-Guided Neural Architecture Evolution. *Bio-inspired Modeling of Cognitive Tasks,* Volume 4527, pp. 437--446.

Darwin C. (1859). *On the origin of the species by means of natural selection.* London: John Murray.

Dasgupta D. a. M. Z. (2013). *Evolutionary algorithms in engineering applications.* Springer Science & Business Media.

DeWitt T. J. Sih A. & Wilson D. S. (1998). Costs and limits of phenotypic plasticity. *Trends in ecology & evolution,* 13(2), pp. 77-81.

Dignum S. & Poli R. (2007). Generalisation of the limiting distribution of program sizes in tree-based genetic programming and analysis of its effects on bloat. *In Proceedings of the 9th annual conference on Genetic and evolutionary computation,* ACM, pp. 1588--1595.

Dong, L.-y.et al., (2007). Classifier learning algorithm based on genetic algorithms. *In Proceedings of ICICIC'07. Second International Conference on Innovative Computing, Information and Control, 2007. ,* IEEE, pp. 126-129.

Dorigo M. Birattari M. & Stutzle T. (2006). Ant colony optimization. *IEEE computational intelligence magazine,* 1(4), pp. 28-39.

Downing K. L. (2001). Genetic Programming and Evolvable Machines. *Genetic Programming and Evolvable Machines,* 2(3), pp. 259-288.

Engelbrecht A. P. (2007). *Computational intelligence: an introduction.* John Wiley & Sons.

Eshelman L. J. & Schaffer J. D. (1992). Real-Coded Genetic Algorithms and Interval-Schemata. *In Foundations of genetic algorithms,* pp. 187-202.

Font J. M. a. Manrique D. a. Ramos-Criado. P. & del Rio D. (2016). Partition based real-valued encoding scheme for evolutionary algorithms. *Natural Computing,* 15(3), pp. 477--492.

Font J. M. Manrique D. & Ríos J. (2009). *Redes de Neuronas Artificiales y Computación Evolutiva.* Madrid: Fundación General de la UPM.

Galván-López E. McDermott J. a. O. M. & Brabazon A. (2011). Defining locality as a problem difficulty measure in genetic programming. *Genetic Programming and Evolvable Machines,* 12(4), pp. 365-401.

Galvan-Lopez E. a. O. M. a. B. A. (2009). Towards understanding the effects of locality in GP. *In Eighth Mexican International Conference on Artificial Intelligence (MICAI),* IEEE, pp. 9-14.

Galvan E. Trujillo L. McDermott J. & Kattan A. (2013). Locality in continuous fitness-valued cases and genetic programming difficulty. *In EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation II,* Springer, pp. 41-56.

García-Arnau M. Manrique D. Rios, Juan & Rodríguez-Patón A. (2007). Initialization method for grammar-guided genetic programming. *Knowledge-Based Systems,* 20(2), pp. 127-133.

Giegerich R. (2014). Introduction to stochastic context free grammars. *RNA Sequence, Structure, and Function: Computational and Bioinformatic Methods,* pp. 85-106.

Goldberg D. E. (1989). *Genetic algorithms in search optimization and machine learning.* Addison-wesley Reading Menlo Park.

Goldberg D. E. & Holland J. H. (1988). Genetic algorithms and machine learning. *Machine learning,* 3(2), pp. 95-99.

Godfrey-Smith P. (2009). *Theory and reality: An introduction to the philosophy of science.* Chicago: University of Chicago Press.

Gottlieb J. & Raidl G. R. (2000). The effects of locality on the dynamics of decoder-based evolutionary search. *In Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation,* Morgan Kaufmann Publishers Inc., pp. 283-290.

Gruau F. (1996). On using syntactic constraints with genetic programming. In: *Advances in Genetic Pro- gramming.* Cambridge(MA): MIT Press, pp. 377-394.

Gwiazda T. D. (2006). *Crossover for single-objective numerical optimization problems.*

Hansen, N. (2006). The CMA evolution strategy: a comparing review. *Towards a new evolutionary computation*, pp.75-102.

Harley C. B. (1981). Learning the evolutionarily stable strategy. *Journal of theoretical biology,* 89(4), pp. 611-633.

Hasegawa Y. & Iba H. (2009). Latent variable model for estimation of distribution algorithm based on a probabilistic context-free grammar. *IEEE Transactions on Evolutionary Computation,* 13(4), pp. 858-878.

Hauschild M. a. P. M. (2011). An introduction and survey of estimation of distribution algorithms. *Swarm and Evolutionary Computation,* 1(3), pp. 111-128.

Haynes T. Schoenefeld D. A. & Wainwright R. L. (1996). Type inheritance in strongly typed genetic programming. *Advances in genetic programming,* 2(2), pp. 359--376.

Heywood M. I. & Lichodzijewski P. (2010). Symbiogenesis as a mechanism for building complex adaptive systems: A review. *In European Conference on the Applications of Evolutionary Computation,* Springer, pp. 51-60.

Hien N. T. & Hoai N. X. (2006). A brief overview of population diversity measures in genetic programming. *In 3rd Asian-Pacific Workshop on Genetic Programming,* Hanoi, Vietnam, pp. 128-139.

Hinton G. E. & Nowlan S. J. (1987). How learning can guide evolution. *Complex systems,* 1(3), pp. 495-502.

Hoai N. X. McKay R. I. & Essam D. (2006). Representation and structural difficulty in genetic programming. *IEEE Transactions on evolutionary computation,* 10(2), pp. 157-166.

Hochberg Z. (2011). *Evo-devo of child growth: treatise on child growth and human evolution.* John Wiley & Sons.

Holland J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* MIT press.

Hopcroft J. E. Motwani R. & Ullman J. D. (2013). *Introduction to automata theory, languages, and computation.* 3rd Edition ed. Pearson.

Hopcroft J. E. Motwani R. & Ullman J. D. (2013). *Introduction to automata theory, languages, and computation.* Pearson.

Hopcroft J. E. & Ullman J. D. (1969). *Formal languages and their relation to automata.* Addison-Wesley Longman Publishing Co., Inc.

Horner H. (1996). *A C++ class library for genetic programming,* Citeseer.

Iba H. (1996). Random tree generation for genetic programming. In: Springer, pp. 144-153.

Isasi P. Martínez P. & Borrajo D. (1997). *Lenguajes, gramáticas y autómatas: un enfoque práctico.* Pearson Educación.

Joshi A. K. & Schabes Y. (1997). Tree-adjoining grammars. In: *Handbook of formal languages.* Springer, pp. 69-123.

Kari L. & Rozenberg G. (2008). The Many Facets of Natural Computing. *Communications of the ACM,* October, 51(10), pp. 72-83.

Keijzer M. (2003). Improving symbolic regression with interval arithmetic and linear scaling. *In European Conference on Genetic Programming,* Springer, pp. 70-82.

Keijzer M. & Babovic V. (1999). Dimensionally aware genetic programming. *In 1st Annual Conference on Genetic and Evolutionary Computation,* Morgan Kaufmann Publishers Inc., pp. 1069--1076.

Kim K. a. M. R. I. (2013). Stochastic diversity loss and scalability in estimation of distribution genetic programming. *IEEE Transactions on Evolutionary Computation,* 17(3), pp. 301-320.

Kim K. Shan Y. Nguyen X. H. & McKay R. I. (2014). Probabilistic model building in genetic programming: a critical review. *Genetic Programming and Evolvable Machines,* 15(2), pp. 115-167.

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

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

Koza, J. R. et al., (2006). *Genetic programming IV: Routine human-competitive machine intelligence.* Springer Science & Business Media.

Krithivasan K. (2009). *Introduction to Formal Languages, Automata Theory and Computation.* 1st Edition ed. Pearson.

Langdon W. B. (1998). *Genetic Programming and Data Structures.* Springer US.

Langdon W. B. (2000). Size fair and homologous tree crossovers for tree genetic programming. *In Genetic programming and evolvable machines,* Springer, pp. 95-119.

Langdon W. B. & Poli R. (1998). Fitness causes bloat. *In Soft Computing in Engineering Design and Manufacturing,* Springer, pp. 13-22.

Langdon W. B. & Poli R. (1998). Why ants are hard. *In Third Annual Conference. Genetic Programming,* Madison (WI), USA, Morgan Kaufmann, pp. 193-201.

Langdon W. B. & Poli R. (2013). *Foundations of genetic programming.* Springer Science.

Langdon W. B. Soule T. Poli R. & Foster J. A. (1999). The evolution of size and shape. *Advances in genetic programming,* Volume 3, pp. 163-190.

Larrañaga P. & Lozano J. A. (2001). *Estimation of distribution algorithms: A new tool for evolutionary computation.* Springer Science & Business Media.

Lichman M. (2013). *UCI Machine Learning Repository.* University of California, Irvine, School of Information and Computer Sciences. http://archive.ics.uci.edu/ml

Luke S. (2000). Two fast tree-creation algorithms for genetic programming. *In IEEE Transactions on Evolutionary Computation,* IEEE, pp. 274-283.

Luke S. & Panait L. (2001). A survey and comparison of tree generation algorithms. *In Genetic and Evolutionary Computation Conference (GECCO-2001),* pp. 81-88.

Margulis L. (1993). Origins of species: acquired genomes and individuality. *BioSystems,* 31(2), pp. 121-125.

Martin J. C. (2011). *Introduction to Languages and the Theory of Computation.* New York(NY): Mc Graw Hill.

Mckay, R. I. et al., (2010). Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines,* 11(3-4), pp. 365-396.

McPhee N. F. & Miller J. D. (1995). Accurate replication in genetic programming. *In ICGA,* Pittsburgh,, Morgan Kaufmann, pp. 303-309.

Michalewicz Z. a. H. S. J. (1996). Genetic Algorithms+ Data Structures= Evolution Programs. *Mathematical Intelligencer,* 18(3), p. 71.

Migdalas A. Pardalos P. M. & Värbrand P. (2013). *Multilevel optimization: algorithms and applications.* Springer Science & Business Media.

Moll R. N. Arbib M. A. & Kfoury A. J. (1988). *An Introduction to Formal Language Theory.* 1st Edition ed. Springer.

Montana D. J. (1995). Strongly typed genetic programming. *Evolutionary computation,* 3(2), pp. 199-230.

Morgan C. L. (1896). On modification and variation. *Science,* pp. 733-740.

Mukherjee S. & Eppstein M. J. (2012). Differential evolution of constants in genetic programming improves efficacy and bloat. *In Proceedings of the 14th annual conference companion on Genetic and evolutionary computation,* ACM, pp. 625-626.

Murphy, E. et al., (2012). Grammar bias and initialisation in grammar based genetic programming. In: *Genetic Programming.* Springer Berlin Heidelberg, pp. 85-96.

Noraini M. & Geraghty J. (2011). Genetic algorithm performance with different selection strategies in solving TSP. *World Congress on Engineering,* Volume 2, pp. 4-9.

Nowling R. J. & Mauch H. (2011). Priority encoding scheme for solving permutation and constraint problems with genetic algorithms and simulated annealing. *In 2011 Eighth International Conference on Information Technology: New Generations (ITNG),* Las Vegas, USA, IEEE, pp. 810-815.

O'Neill M. & Brabazon A. (2006). Grammatical swarm: The generation of programs by social programming. *Natural Computing,* 5(4), pp. 443-462.

O'Neill M. & Brabazon A. (2006). Grammatical Differential Evolution. *In Proceedings of the International Conference on Artificial Intelligence,* Las Vegas, CSEA Press, pp. 2331-236.

O'Neil M. & Ryan C. (2003). Grammatical evolution. In: *Grammatical evolution.* Springer, pp. 33-47.

O'Neill M. Vanneschi L. Gustafson S. & Banzhaf W. (2010). Open issues in genetic programming. *Genetic Programming and Evolvable Machines,* 11(3-4), pp. 339-363.

O'Sullivan J. & Ryan C. (2002). An investigation into the use of different search strategies with grammatical evolution. *In Genetic Programming, Proceedings of the 5th European Conference, EuroGP,* Kinsale, Springer, pp. 268-277.

O'Reilly U.-M. & Oppacher F. (1994). The Troubling Aspects of a Building Block Hypothesis for Genetic Programming. *FOGA*, pp. 73-88.

Osborn H. F. (1896). Ontogenic and phylogenic variation. *Science,* pp. 786-789.

Palma Méndez J. T. & Morales R. M. (2008). *Inteligencia artificial: técnicas, métodos y aplicaciones.* Madrid: McGraw Hill.

Panyaworayan W. & Wuetschner G. (2002). Time series prediction using a recursive algorithm of a combination of genetic programming and constant optimization. *Facta universitatis-series: Electronics and Energetics,* 15(2), pp. 265-279.

dal Piccol Sotto L. F. & de Melo V. V. (2016). Studying bloat control and maintenance of effective code in linear genetic programming for symbolic regression. *Neurocomputing,* Volume 180, pp. 79-93.

Poli R. Langdon W. B. & Dignum S. (2007). On the limiting distribution of program sizes in tree-based genetic programming. *In European Conference on Genetic Programming,* Springer, pp. 193-204.

Poli R. Langdon W. B. McPhee N. F. & Koza J. R. (2008). *A field guide to genetic programming.*

Poli R. (2000). Hyperschema theory for GP with one-point crossover, building blocks, and some new results in GA theory. *Genetic Programming,* pp. 163-180.

Prusinkiewicz P. a. L. A. (2012). *The Algorithmic Beauty of Plants.* Springer Science & Business Media.

Radcliffe N. J. (1991). Equivalence class analysis of genetic algorithms. *Complex systems,* 5(2), pp. 183-205.

Ratle A. & Sebag M. (2000). Genetic programming and domain knowledge: Beyond the limitations of grammar-guided machine discovery. *In Parallel Problem Solving from Nature PPSN VI,* Springer, pp. 211--220.

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

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

Rojas S. A. a. B. P. J. (2004). A grid-based ant colony system for automatic program synthesis. *In 2004 Genetic and Evolutionary Computation Conference,* Seattle, WA, USA, Citeseer.

Russell S. J. & Norvig P. (2009). *Artificial Intelligence: A Modern Approach.* 3rd Edition ed. Prentice Hall.

Ryan C. Collins J. & Neill M. (1998). Grammatical evolution: Evolving programs for an arbitrary language. *In European Conference on Genetic Programming,* Springer, pp. 83--96.

Salustowicz R. & Schmidhuber J. (1997). Probabilistic incremental program evolution. *Evolutionary Computation,* 5(2), pp. 123-141.

Sapp J. Carrapiço F. & Zolotonosov M. (2002). Symbiogenesis: the hidden face of Constantin Merezhkowsky. *History and philosophy of the life sciences,* pp. 413-440.

de Saussure F. (1992). *Course in General Linguistics.* La Salle(Illinois): Open Court.

Shan R. a. M. R. a. A. H. a. E. D. (2003). Program evolution with explicit learning. *In Proceedings of CEC'03. The 2003 Congress on Evolutionary Computation, 2003,* IEEE, pp. 1639-1646.

Shan, Y. a. M. R. I. et al., (2004). Grammar model-based program evolution. *In Proceedings of CEC2004. Congress on Evolutionary Computation, 2004.* , IEEE, pp. 478-485.

Shan Y. McKay R. I. Essam D. & Abbass H. A. (2006). A survey of probabilistic model building genetic programming. In: *Scalable Optimization via Probabilistic Modeling.* Springer, pp. 121-160.

Silva S. & Costa E. (2009). Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories. *Genetic Programming and Evolvable Machines,* 10(2), pp. 141-179.

Sipser M. (2012). *Introduction to the Theory of Computation.* 3rd Edition ed. Cengage Learning.

Sipser M. (2012). *Introduction to the Theory of Computation.* 3rd Edition ed. Cengage Learning.

Sivaraj R. & Ravichandran T. (2011). A review of selection methods in genetic algorithm. *International journal of engineering science and technology,* 1(3), pp. 3792--3797.

Soule T. & Foster J. A. (1998). Removal bias: a new cause of code growth in tree based evolutionary programming. *In Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on,* 1998, IEEE, pp. 781-786.

Su J. Zhang H. Ling C. X. & Matwin S. (2008). Discriminative parameter learning for bayesian networks. *In Proceedings of the 25th international conference on Machine learning,* ACM, pp. 1016-1023.

Tanev I. (2004). Implications of incorporating learning probabilistic context-sensitive grammar in genetic programming on evolvability of adaptive locomotion gaits of snakebot. *In Proceedings of GECCO 2004,* Seattle, Washington, USA,

Topchy A. & Punch W. F. (2001). Faster Genetic Programming based on Local Gradient Search of Numeric Leaf Values. *In Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation,* 2001, Morgan Kaufmann Publishers Inc., pp. 155-162.

Trujillo L. Muñoz L. Silva S. & Galván-López E. (2016). neat Genetic Programming: Controlling bloat naturally. *Information Sciences,* Volume 333, pp. 21-43.

Turney P. Whitley D. & Anderson R. W. (2007). Evolution, learning, and instinct: 100 years of the Baldwin effect. *MIT Press.*

Uy N. Hoai N. O'Neill M. & McKay B. (2010). The role of syntactic and semantic locality of crossover in genetic programming. *Parallel Problem Solving from Nature, PPSN XI,* pp. 533-542.

Uy, N. Q. et al., (2013). On the roles of semantic locality of crossover in genetic programming. *Information Sciences,* Volume 235, pp. 195-213.

Vanneschi L. Castelli M. & Silva S. (2014). A survey of semantic methods in genetic programming. *Genetic Programming and Evolvable Machines,* Volume 15, pp. 195-214.

Vanneschi L. Mauri G. Valsecchi A. & Cagnoni S. (2006). Heterogeneous cooperative coevolution: strategies of integration between GP and GA. *In Proceedings of the 8th annual conference on Genetic and evolutionary computation,* ACM, pp. 361--368.

Ványi R. & Zvada S. (2003). Avoiding syntactically incorrect individuals via parameterized operators applied on derivation trees. *In Evolutionary Computation,* pp. 2791--2798.

Whigham P. A. (1995). A schema theorem for context-free grammars. *In Evolutionary Computation, IEEE International Conference on,* p. 178.

Whigham P. A. (1995). Grammatically-based genetic programming. *In Workshop on Genetic Programming: From Theory to Real-World Applications,* Tahoe City, California, USA, J. P. Rosca, p. 33–41.

Whigham P. A. (1995). Inductive bias and genetic programming. pp. 461-466.

Whigham P. A. (1995). Inductive bias and genetic programming. *In First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications,* GALESIA, Sheffield, UK, p. 461-466.

White, D. R. et al., (2013). Better GP benchmarks: community survey results and proposals. *Genetic Programming and Evolvable Machines,* 14(1), pp. 3-29.

Whitley D. Gordon V. & Mathias K. (1994). Lamarckian evolution, the Baldwin effect and function optimization. *Parallel Problem Solving from Nature—PPSN III,* pp. 5-15.

Wischmann S. Stamm K. & Wörgötter F. (2007). Embodied evolution and learning: The neglected timing of maturation. *Advances in Artificial Life,* pp. 284-293.

Wright A. H. (1991). Genetic algorithms for real parameter optimization. *Foundations of genetic algorithms,* Volume 1, pp. 205-218.

Yu T. Riolo R. & Worzel B. (2006). *Genetic Programming Theory and Practice III.* Springer Science & Business Media.