Jessica Mégane Taveira da Cunha

PROBABILISTIC GRAMMATICAL EVOLUTION

1 2 9 0 UNIVERSIDADE D COIMBRA

UNIVERSIDADE D
COIMBRA

Jessica Mégane Taveira da Cunha

PROBABILISTIC GRAMMATICAL EVOLUTION

Dissertation in the context of the Master in Informatics Engineering, Specialization in Intelligent Systems, supervised by Professor Nuno Lourenço and Penousal Machado presented to the Faculty of Sciences and Technology / Department of Informatics Engineering.

September 2021

Faculty of Sciences and Technology

Department of Informatics Engineering

Probabilistic Grammatical Evolution

Jessica Mégane Taveira da Cunha

Dissertation in the context of the Master in Informatics Engineering, Specialization in Intelligent Systems, supervised by Professor Nuno Lourenço and Professor Penousal Machado and presented to the
Faculty of Sciences and Technology / Department of Informatics Engineering.

September 2021

UNIVERSIDADE D
COIMBRA

This page is intentionally left blank.

Abstract

Grammatical Evolution (GE) [1] is one of the most popular variants of Genetic Programming (GP) [2] and has been successfully used in a wide range of problem domains. Since the original proposal, many improvements have been introduced in GE to improve its performance by addressing some of its main issues, namely low locality and high redundancy [3, 4].

In grammar-based GP methods the choice of the grammar has a significant impact on the quality of the generated solutions, since it is the grammar that defines the search space [5]. In this work, we present four variants of GE, which during the evolutionary process perform an exploration of the search space by updating the weights of each rule of the grammar. These variants introduce two alternative representation types, two grammar adjustment methods, and a new mapping method using a Probabilistic Context-Free Grammar (PCFG).

The first method is Probabilistic Grammatical Evolution (PGE), in which individuals are represented by a list of real values (genotype), each value denoting the probability of selecting a derivation rule. The genotype is mapped into a solution (phenotype) to the problem at hand, using a PCFG. At each generation, the probabilities of each rule in the grammar are upated, based on the expansion rules used by the best individual. Co-evolutionary Probabilistic Grammatical Evolution (Co-PGE) employs the same representation of individuals and introduces a new technique to update the grammar's probabilities, where each individual is assigned a PCFG where the probabilities of each derivation option are changed at each generation using a mutation like operator. In both methods, the individuals are remapped after updating the grammar.

Probabilistic Structured Grammatical Evolution (PSGE) and Co-evolutionary Probabilistic Structured Grammatical Evolution (Co-PSGE) were created by adapting the mapping and probabilities update mechanism from PGE and Co-PGE to Structured Grammatical Evolution (SGE), a method that was proposed to overcome the issues of GE while improving its performance [6]. These variants use as genotype a set of dynamic lists, one for each non-terminal of the grammar, with each element of the list being the probability used to map the individual with the PCFG.

We analyse and compare the performance of all the methods in six benchmarks. When compared to GE, the results showed that PGE and Co-PGE are statistically similar or better on all problems, while PSGE and Co-PSGE are statistically better on all problems. We also highlight Co-PSGE since it is statistically superior to SGE in some problems, making it competitive with the state-of-the-art. We also performed an analysis on the representations, and the results showed that PSGE and Co-PSGE have less redundancy, and PGE exhibited better locality than GE, which allows for a better exploration of the search space.

The analyses conducted showed that the evolved grammars help guide the evolutionary process and provides us information about the most relevant production rules to generate better solutions. In addition, they can also be used to generate a sampling of solutions with better average fitness.

Keywords

Genetic Programming, Grammatical Evolution, Probabilistic Context-Free Grammar, Genotype-Phenotype Mapping, Co-evolution.

This page is intentionally left blank.

Resumo

Evolução Gramatical (GE) [1] é uma das variantes mais populares de Programação Genética (GP) [2] e tem sido utilizada com sucesso em problemas de vários domínios. Desde a proposta original, muitas melhorias foram introduzidas na GE para melhorar a sua performance abordando alguns dos seus principais problemas, nomeadamente a baixa localidade e a alta redundância [3, 4].

Nos métodos de GP baseados em gramáticas, a escolha da gramática tem um papel importante na qualidade das soluções geradas, uma vez que é a gramática que define o espaço de procura [5]. Neste trabalho, propomos quatro variantes da GE, que durante o processo evolucionário realizam uma exploração do espaço de procura alterando os pesos de cada regra da gramática. Estas variantes introduzem dois tipos de representação alternativas, dois métodos diferentes de ajustar a gramática e um novo método de mapeamento, utilizando uma Gramática Livre de Contexto Probabilistica (PCFG).

O primeiro método é a Evolução Gramatical Probabilistica (PGE), no qual os indivíduos são representados por uma lista de probabilidades (genótipo), onde cada valor representa a probabilidade de selecionar uma regra de derivação. O genótipo é mapeado numa solução para o problema em questão (fenótipo), recorrendo a uma PCFG. A cada geração, as probabilidades de cada regra da gramática são atualizadas, com base nas regras de expansão usadas pelo melhor indivíduo. A Evolução Gramatical Probabilistica Co-Evolucionária (Co-PGE) utiliza a mesma representação dos indivíduos e introduz uma nova técnica de atualização das probabilidades da gramática onde as probabilidades de cada regra de derivação são alteradas a cada geração usando um operador semelhante à mutação. Em ambos os métodos os indivíduos são remapeados após atualização da gramática.

A Evolução Gramatical Estruturada Probabilistica (PSGE) e a Evolução Gramatical Estruturada Probabilistica Co-Evolucionária (Co-PSGE) foram criadas adaptando a Evolução Gramatical Estruturada (SGE), um método que foi proposto para superar os problemas da GE melhorando a sua performance [6]. Estas variantes usam como genótipo um conjunto de listas dinâmicas, uma para cada não-terminal, em que cada elemento da lista é uma probabilidade usada para mapear o indivíduo, usando uma PCFG.

Analisamos e comparamos o desempenho dos métodos em seis problemas benchmark. Quando comparados com a GE, os resultados mostraram que a PGE e a Co-PGE são estatisticamente semelhantes ou melhores em todos os problemas, enquanto que a PSGE e a Co-PSGE foram estatisticamente melhores em todos os problemas do que a tradicional GE. Destacamos também a Co-PSGE por superar estatisticamente a SGE em alguns problemas, tornando-a competitiva com o estado da arte. Também realizamos uma análise nas representações, e os resultados mostraram que a PSGE e a Co-PSGE tem menos redundancia, e o PGE apresenta localidade mais elevada que o GE, o que permite uma melhor exploração do espaço de procura.

As análises efetuadas mostraram que as gramáticas evoluidas ajudam a guiar o processo evolucionario, e fornecem-nos informações sobre as regras de produção mais relevantes para gerar melhores soluções. Além disso, também podem ser utilizadas para gerar uma amostragem de soluções com melhor fitness médio.

Palavras-Chave

Programação Genética, Evolução Gramatical, Gramática Livre de Contexto Probabilistica, Mapeamento Genótipo-Fenótipo, Co-evolução.

This page is intentionally left blank.

Acknowledgements

I would like to thank my supervisors, Professor Nuno Lourenço and Professor Penousal Machado for their guidance, support and the opportunity to work with them. Professor Penousal, I never told you but I became a fan of yours in the Introduction to Artificial Intelligence classes, so it was an honour for me when you joined this project. I learned a lot from you, and for that, thank you! Professor Nuno, by talking to you I made the decision to work with Artificial Intelligence for the rest of my life. When I entered the Masters I had the opportunity to do a three-month fellowship with you, and start this project. I never thought that two years later this decision would lead me to have a published paper and a thesis written, and a whole potential for improvement ahead of me. You taught me what it is to be a scientist, and especially how to deal when the results are not what I expected. Thank you for your availability, and above all, for your patience.

To my parents and sister, thank you for everything. You still ask me "After all, what is the purpose of what you are doing?" and even if you still do not understand, you never doubted and are always the first to support me. Mom and Dad, you have worked hard all your lives to give us the opportunity to study, and for that I will be eternally grateful, and it is my mission to make you proud.

Victor, Catarina, Marco, Isabel and Eduardo, thank you for being my "rubber ducks" and for all the emotional support you give. Thank you for listening to me when things are not going so well, for listening to me when things are going well, for always being willing to help, for the random revisions to the thesis that I asked for, and especially for making me leave the house once in a while. Without you these last years would have been much harder to get through.

I would also like to thank my childhood friends from Coura, the ones I made in Coimbra and Emily, you may not realize it but your support and friendship helped me to keep my spirits up and mentally sane.

This work was partially funded by the project grant DSAIPA/DS/0022/2018 (GADgET) and by national funds through the FCT - Foundation for Science and Technology, I.P., within the scope of the project CISUC - UID/CEC/00326/2020 and by European Social Fund, through the Regional Operational Program Centro 2020. We also thank the NVIDIA Corporation for the hardware granted to this research.

This page is intentionally left blank.

Contents

This page is intentionally left blank.

Acronyms

πGE Position Independent Grammatical Evolution. 12, 14

(GE)^2 Grammatical Evolution by Grammatical Evolution. 14

ADF Automatically Defined Functions. 9

AI Artificial Intelligence. 5

CDT Conditional Dependency Tree. 13

CFG Context-Free Grammar. 1, 10, 11

CFG-GP Context-Free Grammar Genetic Programming. 1, 9, 10

Co-PGE Co-evolutionary Probabilistic Grammatical Evolution. iii, 3, 22, 25, 34, 36, 40–43, 45, 51–54, 63, 64

Co-PSGE Co-evolutionary Probabilistic Structured Grammatical Evolution. iii, 3, 33, 36, 40–43, 45, 51–54, 63, 64

DSGE Dynamic Structured Grammatical Evolution. 14, 15

EA Evolutionary Algorithm. 1, 5, 6, 8

EC Evolutionary Computation. 5

EDA Estimation Distribution Algorithm. 13

GE Grammatical Evolution. iii, 1, 2, 9–15, 19, 22, 23, 25, 29, 36, 40–43, 49, 51–56, 58–60, 63, 64

GP Genetic Programming. iii, 1, 8–11, 13, 15, 20, 37, 44, 63

mGGA meta-Grammar Genetic Algorithm. 14

MI Mutation Innovation. 18, 19, 54, 55, 59, 60

PCFG Probabilistic Context-Free Grammar. iii, 2, 3, 13, 22, 23, 25, 26, 29, 30, 32–34, 44, 45, 47, 51, 55, 56, 60, 63, 64

PGE Probabilistic Grammatical Evolution. iii, 2, 22, 25, 26, 29, 30, 32, 36, 40–45, 49–60, 63, 64

PI Grow Position Independent Grow. 14

PMBGE Probabilistic Model Building Grammatical Evolution. 13

PSGE Probabilistic Structured Grammatical Evolution. iii, 2, 25, 29, 32, 33, 36, 40–44, 51–60, 63, 64

PTC2 Probabilistic Tree Creation 2. 14

RRSE Root Relative Squared Error. 36

SGE Structured Grammatical Evolution. iii, 2, 3, 12–15, 17, 29, 30, 33, 36, 40–43, 51–54, 56–60, 63, 64

TAG Tree-Adjunct Grammar. 13

TAGE Tree-Adjunct Grammatical Evolution. 13

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

Chapter 1

Introduction

Evolutionary Algorithms (EAs) is the name given to a set of stochastic search procedures that are loosely inspired by the principles of natural selection and genetics. These methods iteratively improve a set of candidate solutions, usually referred to as the population, guided by an objective function. The improvement is obtained by selecting the most promising solutions (taking into account the objective function), and applying some stochastic variations using operators similar to mutations and recombinations that take place in biological reproduction. Individuals with higher fitness are more likely to survive and reproduce. The application of these elements is repeated for several generations and it is expected that, over time, the quality of individuals improves.

Genetic Programming (GP) [2] is a branch of EAs used for the automatic evolution of programs. Over the years many variants of GP have been proposed, namely concerned with how individuals (i.e., computer programs) are represented. Some of these variants make use of grammars to enforce syntactic constraints on the individual solutions. The most well known grammar-based GP approaches are Context-Free Grammar Genetic Programming (CFG-GP), introduced by Whigham [7], and Grammatical Evolution (GE) introduced by Ryan et al. [1, 8]. The main distinction between the two approaches is the representation of the individual's solution in the search space. CFG-GP uses a derivation-tree based representation, and the mapping is made by reading the terminal symbols (tree leaves), starting from the left leaf to the right. In GE the individuals are represented as a variable-length string of integers (i.e., the genotype), which are mapped into the solution of the problem (i.e., phenotype). The mapping resorts to a Context-Free Grammar (CFG) in the genotype-phenotype mapping to specify the syntax and search space of the solutions. Each value of the genotype (i.e., codon) is used to choose a production rule of the grammar until it forms a valid program.

GE is one of the most widely used GP variants, but it presents some problems, namely low locality and high redundancy [3, 4]. The redundancy corresponds to the existence of several genotypes that represent the same phenotype and the locality describes how changes in the genotype affect the phenotype. GE suffers from low locality because small changes in the genotype often cause large changes in the phenotype. This causes exploitation to be exchanged for exploration, which can lead to a behaviour similar to random search [9].

To overcome some of the main criticisms of GE, numerous methods have been proposed in the literature. Most of these methods perform changes in grammars [5, 10, 11, 12], representation of individuals [13, 14, 15, 16, 17, 18] or population initialization [19, 20, 21, 22, 23].

One of the approaches that has proven to be effective in addressing the locality and redundancy issues of GE is Structured Grammatical Evolution (SGE) [6, 24]. SGE uses dynamic lists of ordered integers as genotype of individuals, with one list for each non-terminal of the grammar used to perform the mapping. Each value in the list (i.e., codon) represents which production rule to choose from the respective non-terminal. This representation allows genotypic changes to not affect the production rules chosen by the other non-terminals, resulting in a higher locality and lower redundancy, while achieving a better performance in fewer generations [16, 25].

1.1 Motivation

When one wants to apply GE to a problem, it is necessary to design a grammar and the fitness function to evaluate the solutions. The choice of grammar has a big impact on the algorithm's behaviour, because it constrains the search space of possible solutions by restricting it to its syntax [5]. The motivation for this work arose in the interest of creating a probabilistic version of GE that throughout the evolutionary process would conduct a biased exploration of the search space, adjusting the weights of each symbol of the grammar.

We hypothesise that a biased grammar may serve a purpose other than simply guide the evolution. By analysing the probabilities assigned to each production rule of an evolved grammar for a specific problem one could potentially have a better comprehension of the problem, as well as serve as a blueprint for generating individuals with better fitness in fewer generations. It would also be interesting to know if a grammar, resulting from the evolution of a population within a specific problem, could be used in other problems of the same family.

1.2 Contributions

This work resulted in four different contributions, which resort to a Probabilistic Context-Free Grammar (PCFG) to perform the mapping between the genotype and phenotype. The novelty of these algorithms lies in the new representation introduced and in the way the probabilities of the grammar evolves over generations. We start by proposing two variants of GE which were subsequently adapted to SGE, creating two more new methods. The main contributions can be summarised as follows:

- Probabilistic Grammatical Evolution (PGE) is a new GE variant in which at the end of each generation the probabilities of the PCFG are updated based on the phenotype of a selected individual and a learning factor. The updated grammar is used to map the individuals of the next generation. This method was published and presented at EuroGP 2021 [18], one of the most prestigious conferences from the Genetic Programming field. We also have the source code of this method publicly available on GitHub[1].

- Probabilistic Structured Grammatical Evolution (PSGE) is a variant of SGE, that combines the representation of individuals and the mapping of SGE and PGE, using the same probability update mechanism introduced.

[1] `https://github.com/jessicamegane/pge`

- Co-evolutionary Probabilistic Grammatical Evolution (Co-PGE) is a method in which each individual co-evolve a PCFG. In the initialization, each individual is constituted by its genotype and a PCFG with each production having equal probability of being chosen. At each generation individuals suffer variation as well as their grammar which can be mutated to change the probabilities of each derivation option.
- Co-evolutionary Probabilistic Structured Grammatical Evolution (Co-PSGE) is a variant of Co-PGE, that uses the same mechanism of co-evolution proposed, adapted to the representation of individuals used by SGE.

All methods are evaluated in terms of performance, locality and redundancy, following the frameworks proposed by Whigham et al. [9], Raidl et al. [26], and Lourenço et al. [16]. Afterwards, an analysis of the evolution of probabilities in different problems is performed.

1.3 Structure

The remainder of this thesis is structured as follows: Chapter 2 presents the background necessary to understand the work presented, introducing the basics of Evolutionary Computation and GE, as well as related work. Furthermore, validation metrics are also presented. Chapter 3 and 4 present the four proposed methods, detailing the representation and mapping methods used by each. Chapter 5 details the experimentation framework used, as well as the experimental results regarding performance, locality and redundancy. Additionally, an analysis of the evolution of the probabilities of the grammars for the different problems is performed. Chapter 6 gathers the main conclusions and provides some insights regarding future work.

This page is intentionally left blank.

Chapter 2

Background

2.1 Evolutionary Computation

Evolutionary Computation (EC) is an Artificial Intelligence (AI) branch that is inspired by the biological processes of natural evolution. The analogy followed is that evolution starts with a population of individuals with each individual representing a candidate solution and having different capabilities of survival and reproduction. To measure the quality of the solutions, each individual is evaluated with a fitness function, specific to the domain problem. Individuals with better fitness have more chances for survival and are the most suited to breed and create the next generation.

2.1.1 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are metaheuristic algorithms driven by an objective function that follows a trial-and-error approach to problem-solving. The goal is to explore possible solutions in the search space in a stochastic way and find the best solution possible to the problem at hand. Although EAs do not guarantee that they will find the best solution, they can find near-optimal solutions to complex problems.

Darwin's theory of evolution [27] explains the differences between individuals in the same population as a result of natural selection, where the most capable individuals survive through competition between individuals and genetic changes, such as mutation and crossover. These processes have been adapted to computational models called EAs, to simulate the natural behavior of evolution. Alg. 1 shows the main structure of an EA.

The algorithm begins by generating the initial population of individuals (i.e., candidate solutions). Each individual's genetic representation is randomly generated and consists of a collection of codons that form the genotype. The genotype is mapped into the phenotype which represents a possible solution to the problem at hand. The choice of representation is directly linked to the search space. Different EAs can have different genotypic representations, such as binary strings or real values strings.

To assess the performance of each individual on the given problem it is necessary to design a fitness function. This function plays an important role as it evaluates quantitatively the quality of the solutions. The fitness value determines how close the solution is to the objective and is used to compare individuals. Typically, if the genotype does not provide a valid solution to the problem, the fitness function offer a penalty in the quality of the

Algorithm 1 Simple Evolutionary Algorithm

```
1: procedure MAIN(population_size, prob_crossover, prob_mutation, max_generations)
2:     population = initialize_population(population_size)
3:     generation = 0
4:     while generation < max_generations do
5:         evaluate_population(population)
6:         new_population = [ ]
7:         while size(new_population) < population_size do
8:             ind = select_individual(population)
9:             if random() < prob_mutation then
10:                ind = mutate(ind)
11:            end if
12:            if random() < prob_crossover then
13:                ind2 = select_individual(population)
14:                ind = crossover(ind, ind2)
15:            end if
16:            new_population.append(ind)
17:        end while
18:        population = new_population
19:        generation += 1
20:    end while
21:    return best_individual(population)
22: end procedure
```

individual.

To introduce diversity in the population, to escape from local optima, and explore the search space, EAs adapted biological evolutionary processes, such as selection, crossover, and mutation.

It is possible to divide the selection process into parent selection and survivor selection. Parent selection is used to choose individuals to reproduce, generating new individuals for the next generation. This is a stochastic and fitness-biased strategy since individuals with better fitness are more likely to be chosen, but it does not prevent the selection of worse individuals, making it more likely that the offspring will have good characteristics in the genotype. The most used selection methods are the tournament selection and roulette wheel [28]. In tournament selection (Alg. 2), a sample of individuals is randomly picked from the population, and the individual with the highest fitness is selected. In the roulette wheel (Alg. 3), each individual has a probability of being selected proportional to his fitness.

Algorithm 2 Tournament Selection Pseudo-code

```
1: procedure TOURNAMENTSELECTION(population, tournament_size)
2:     tournament_pool = random(population, tournament_size)
3:     sort(tournament_pool)
4:     return tournament_pool[0]
5: end procedure
```

Survivor selection occurs after the offspring is generated. This is the method used to choose which individuals to keep for the next generation. Although the selection can be

Algorithm 3 Roulette Wheel Selection Pseudo-code

```
1: procedure rouletteSelection(population)
2:     sum = 0
3:     cumulative_sum = 0
4:     for individual in population do
5:         sum = sum + individual.fitness
6:     end for
7:     for individual in population do
8:         individual.probability = cumulative_sum + individual.fitness / sum
9:         cumulative_sum = individual.probability
10:    end for
11:    r = random(0,1)
12:    for individual in population do
13:        if individual.probability > r then
14:            return individual
15:        end if
16:    end for
17: end procedure
```

made from various criteria, the most used are based on fitness or age [28]. A fitness-based approach is elitism, which passes a certain number of the best individuals to the next generation, allowing the best solutions not to be lost and increasing the chances of good features appearing in the offspring in the following generations. In the age-based approach, each individual has a maximum limit of generations to exist. This method does not take into account the value of fitness, so the individual can be eliminated even if he has higher fitness.

The individuals chosen by the selection method are used to create descendants and/or mutate. One of the variation operators is crossover. In crossover we take evolved individuals, called parents, and their genetic material is exchanged, taking into account one or more random points of the genotype [28]. The crossover can produce one or two children, but because it is stochastic, if the execution of crossover is repeated with the same parents several times, it does not generate the same offspring. The objective of the crossover is to try to pass the best features, i.e., codons, to the offspring. Fig. 2.1 shows an example of one-point crossover on binary genotypes, producing two offspring.

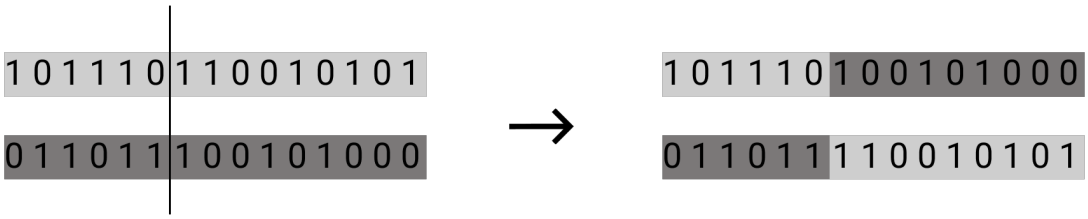


Figure 2.1: Example of one-point crossover generating two offspring.

The mutation operator is applied to an individual and causes changes directly to the genotype, introducing new genetic material. There are several types of mutation and these also vary according to the representation of the genotype. The mutation most used for binary strings is the bitwise mutation [28]. In this type of mutation, one or more codons are randomly chosen to flip the bit. An example of bitwise mutation is represented in Fig. 2.2, where only one codon with value 0 was chosen to mutate, and changed to 1.

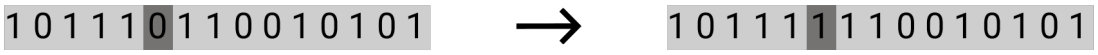


Figure 2.2: Example of bitwise mutation.

Both mutation and crossover have a probability of application associated. A random number is generated, and if it is smaller than the probability, the mutation or crossover occurs. The probability of mutation is usually a low value (ranging from 1% to 10%), since it is applied to each gene of the genotype, and the probability of crossover is usually a high value (ranging from 80% to 100%) since it helps in the exploitation of solutions.

The algorithm is repeated until the number of generations previously established has been reached, and the individual with the best fitness is chosen as the solution.

2.1.2 Genetic Programming

Genetic Programming (GP) [2] is an EA with a particular way of representing the individuals, which is syntax trees. This type of algorithm allows computers to evolve programs to solve problems without the need to program the solution explicitly.

As in EAs, the algorithm is based on the theory of evolution, in which a population is initially generated to create viable solutions to a given problem, represented by the individuals of the population. In GP, the genotype of the individuals is a syntax tree that represents a computer program. The tree nodes represent functions and the leaves represent constants, also called terminals. The functions and terminals used are defined according to the domain of the problem. An example of a GP tree for the function set $F = \{+, -, *, /\}$ and the terminal set $T = \{x, y, 1\}$ is depicted in Fig. 2.3.

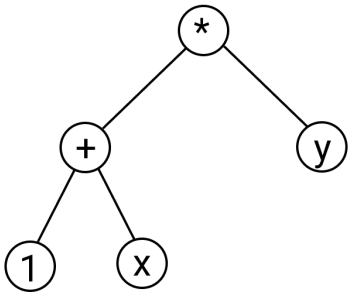


Figure 2.3: Example of syntax tree.

The first startup methods for individuals that have appeared in the literature are the full and the grow methods. First, a depth limit must be defined. This value defines the maximum size of the branches, i.e., the maximum number of edges possible to reach from the root of the tree to any node. The full method (Alg. 4) generates whole trees, and the nodes are chosen randomly within the set of functions until the depth limit is reached. This way, all the leaves have the same depth. The grow method (Alg. 5) allows different distances to the leaves since the nodes are chosen within the set of functions and terminals.

The most popular method of generating trees is the ramped half-and-half [29, 30], which is a combination of full and grow approaches. Usually, a depth limit range is used, and for each value, half of the initial population is created using the full method and half using the grow method, creating different tree sizes.

The fitness of the individuals is usually the error produced by the program in a set of

Algorithm 4 Full Method Pseudo-code

```
1: procedure FULLMETHOD(function_set, terminal_set, depth_limit)
2:     if depth_limit == 0 or random() < |terminal_set| / (|terminal_set|+|function_set|) then
3:         expr = choose_random(terminal_set)
4:     else
5:         func = choose_random(function_set)
6:         expr = (func)
7:         for i = 1 in range(arity(func)) do
8:             expr.append(fullMethod(function_set, terminal_set, depth_limit - 1))
9:         end for
10:    end if
11:    return expr
12: end procedure
```

Algorithm 5 Grow Method Pseudo-code

```
1: procedure GROWMETHOD(function_set, terminal_set, depth_limit)
2:     if depth_limit == 0 then
3:         expr = choose_random(terminal_set)
4:     else
5:         func = choose_random(function_set)
6:         expr = (func)
7:         for i = 1 in range(arity(func)) do
8:             expr.append(growMethod(function_set, terminal_set, depth_limit - 1))
9:         end for
10:    end if
11:    return expr
12: end procedure
```

examples. The best programs are those with fitness closest to zero.

Because the representation is a syntax tree, genetic operators are implemented differently. To perform the crossover, two nodes are randomly chosen from each parent. The descendant is generated by merging a sub-tree of each parent. In Fig. 2.4 it is visible an example of sub-tree crossover. We can notice that a sub-tree has root in one of the chosen nodes, and the other sub-tree has root in the main tree root. Although it is possible to generate two children, the most common in GP is to generate only one [29].

The mutation is simulated by randomly choosing a point from the individual's tree and changing the sub-tree whose root is the selected point, for a new randomly generated one. Fig. 2.5 shows an example of a sub-tree mutation.

Two main problems of GP are modularity [29, 31, 32] and constraints [29]. To help with these issues, some GP variants and improvements have been introduced over the years. Automatically Defined Functions (ADF) was proposed by Koza in [33] and is the most popular modularization method used [29, 31]. To incorporate syntactic constraints, some methods use grammars in the representation of individuals.

The most well known grammar-based GP approaches are Context-Free Grammar Genetic Programming (CFG-GP), introduced by Whigham [7], and Grammatical Evolution (GE) introduced by Ryan et al. [1, 8, 34]. The main distinction between the two approaches is the representation of the individual's solution (genotype) in the search space. Fig. 2.6 shows the representation of the same individual in both approaches, using Grammar 2.1.

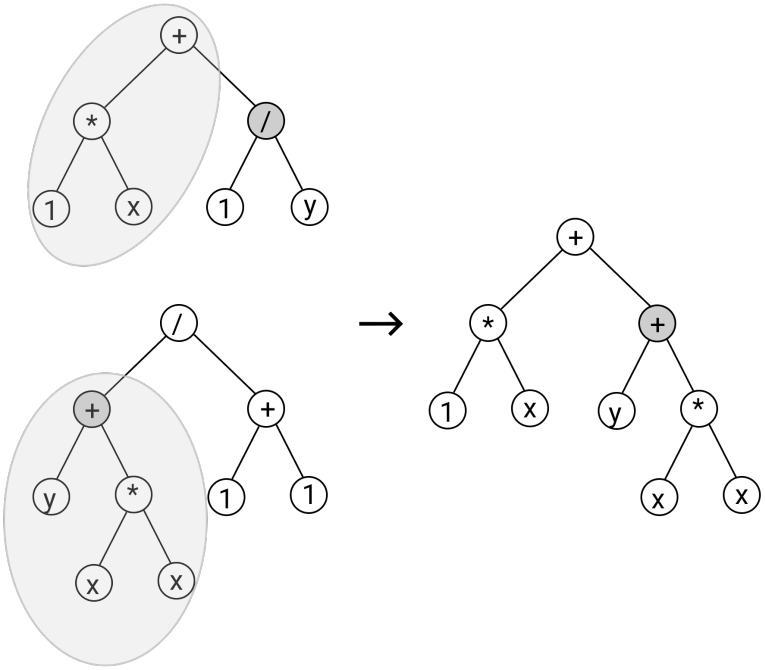


Figure 2.4: Example of sub-tree crossover generating one offspring.

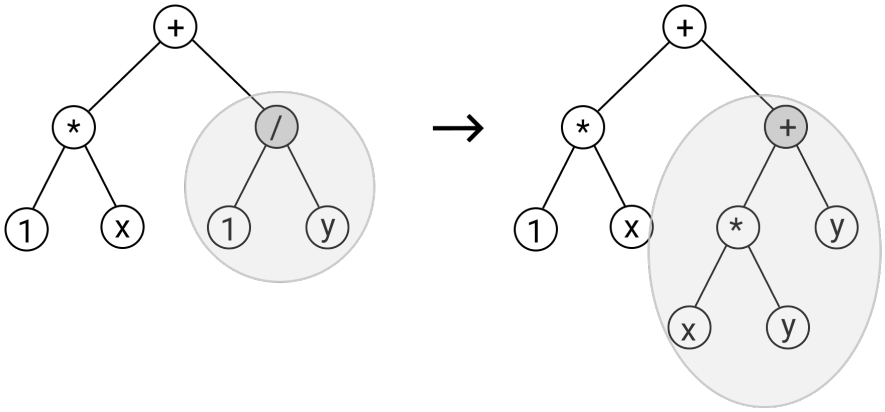


Figure 2.5: Example of sub-tree mutation.

CFG-GP uses a derivation-tree based representation (Fig. 2.6a), and the mapping is made by reading the terminal symbols (tree leaves), starting from the left leaf to the right. In GE there is a distinction in the individual's representation. The individuals are represented as a variable-length string of integers (i.e., genotype), which are mapped into the solution of the problem (i.e., phenotype) (Fig. 2.6b). In this example, it was not necessary to use all codons of the genotype to generate a valid individual. The mapping between the genotype and the phenotype is performed through a Context-Free Grammar (CFG).

There is some debate in the literature regarding the performance of GE compared to other grammar-based variants [9], however GE has become one of the most famous GP variants.

2.1.3 Grammatical Evolution

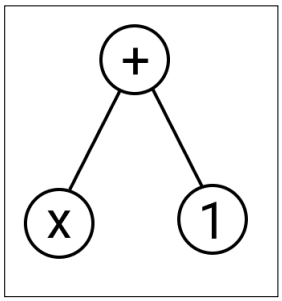
GE [1, 8, 34] is a grammar-based GP approach where the individuals are presented as a variable length string of integers. To create an executable program, the genotype (i.e., the

$\langle expr \rangle$ ::= $\langle expr \rangle$ $\langle op \rangle$ $\langle expr \rangle$ | $\langle var \rangle$

$\langle op \rangle$ ::= + | -

$\langle var \rangle$ ::= x | y | 1.0

Grammar 2.1: Example of Context-Free Grammar.



(a) CFG-GP

Genotype: [62, 33, 54, 88, 13, 142, 25, 202]

<expr> → <expr><op><expr>	$62 mod 2 = 0$
<expr><op><expr> → <var><op><expr>	$33 mod 2 = 1$
<var><op><expr> → x <op><expr>	$54 mod 3 = 0$
x<op><expr> → x + <expr>	$88 mod 2 = 0$
x + <expr> → x + <var>	$13 mod 2 = 1$
x + <var> → x + 1.0	$142 mod 3 = 1$

Phenotype: $x + 1.0$

(b) GE

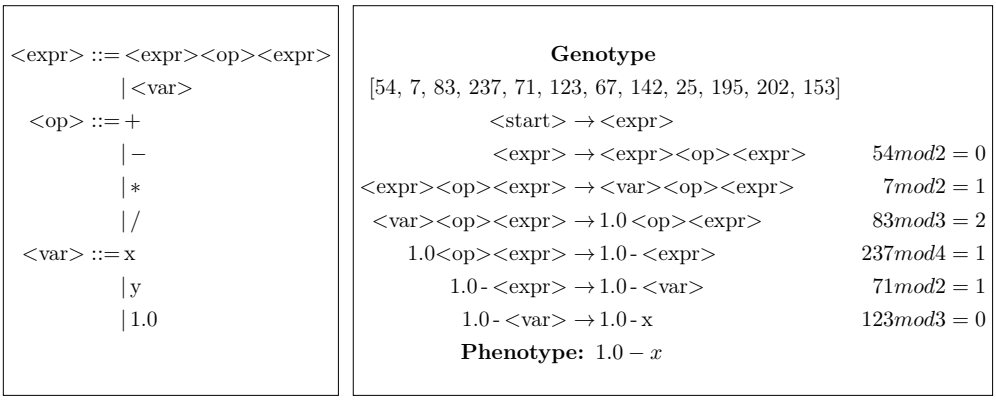
Figure 2.6: Example of representation of individuals.

string of integers) is mapped to the phenotype (i.e., program) via the productions rules defined in a CFG. A grammar is a tuple $G = (NT, T, S, P)$ where NT and T represent the non-empty set of *Non-Terminal (NT)* and *Terminal (T)* symbols, S is an element of NT called the axiom, in which the derivation sequences start, and P is the set of production rules. The rules in P are in the form $A ::= \alpha$, with $A \in NT$ and $\alpha \in (NT \cup T)^*$. The NT and T sets are disjoint. Each grammar defines a language $L(G) = \{w : S \overset{*}{\Rightarrow} w,\ w \in T^*\}$, that is the set of all sequences of terminal symbols that can be derived from the axiom. $*$ represents the unary operator Kleene star.

Similar to how genetics works in nature, the representation of the individual is separated into genotype and phenotype. This approach allows genetic operators to be applied directly to the genotype. The genotype-phenotype mapping is the key issue in GE, and it is performed in several successive steps. In concrete, to select which derivation rule should be selected to replace a non-terminal, the mapping relies on the modulo operator (*mod*). An example of this process is shown in Fig. 2.7, with the left panel showing an example of grammar (Fig. 2.7a), and the right panel showing the mapping process (Fig. 2.7b), using the genotype and grammar example.

The genotype is composed of integers values (i.e., codons) randomly generated in the interval $[0, 255]$. The mapping starts with the axiom, in this case, $< expr >$. Since this non-terminal has two possible expansion rules available, we need to select which one will be used. We start by taking the first unused value of the genotype, which is 54, and divide it by the number of possible options. The remainder of this operation indicates the option that should be used. In our example, $54 mod 2 = 0$, which results in the first production being selected. This process is performed until there are no more non-terminal symbols to expand or there are no more integers to read from the genotype.

In this last case and if we still have non-terminals to expand, a wrapping mechanism can be used, where the genotype will be re-used until it generates a valid individual or the predefined number of wraps is over. If after all the wraps we still have not mapped



(a) Example of Grammar. (b) Genotype-phenotype mapping.

Figure 2.7: Example of the genotype-phenotype mapping of GE.

all the non-terminals, the mapping process stops, and the individual will be considered invalid. The phenotype of each individual is evaluated with the fitness function and then the population goes through the selection mechanisms.

2.2 Related Work

In this section are presented some variants of GE and their improvements. Despite its popularity, GE has been criticized for having high redundancy [4] and low locality [3]. A representation has high redundancy when several different genotypes correspond to one phenotype. Locality is concerned with how changes in the genotype are reflected on the phenotype. These criticisms have triggered many researchers into looking how GE could be improved [6, 13, 15, 17, 35].

2.2.1 Representation Variants

O'Neill et al. [13] proposed Position Independent Grammatical Evolution (πGE), introducing a different mapping mechanism that removes the positional dependency that exists in GE. In πGE each codon is composed of two values (*nont*, *rule*), where *nont* is used to select the next non-terminal to be expanded and the *rule* selects which production rule should be applied from the selected non-terminal. In Fagan et al. [36] several different mapping mechanisms were compared, and πGE showed better performance over GE, with statistical differences.

Another attempt to make GE position independent is Chorus [17]. In this variant, each gene encodes a specific grammar production rule, not taking into consideration the position. This proposal did not show significant differences when comparing with standard GE.

Structured Grammatical Evolution (SGE) [6] is a recent proposal that tackles the locality and redundancy issues of GE [16], at the same time achieving better performance results [37]. SGE proposes an one-to-one mapping between the genotype and the non-terminal symbols of the grammar. Each position in the genotype of SGE is a list of integers associated with a non-terminal, where each element of this list is used to select the next derivation rule of the respective non-terminal. This genotype structure, ensures that the modification of a codon does not affect the chosen productions of other non-terminals, reducing the overall changes that can occur at the phenotypical level, which results in higher

locality.

In [37] different grammar-based GP approaches were compared, and the authors showed that SGE achieved a good performance when compared with several grammar-based GP representations. These results were in line with Fagan et al. [36], which showed that different genotype-phenotype mapping mechanisms can improve the performance of grammar-based GP.

Some GP probabilistic methods have been proposed to try to understand more about the distribution of fitter individuals and have been effective in solving complex problems [38]. Despite its potential, fewer attempts have been made to use probabilities in GE.

Kim et al. [14] implemented a Probabilistic Context-Free Grammar (PCFG) to do the mapping process of GE, where the genotype of the individual is a vector of probabilities used to choose the productions rules. This approach also implements Estimation Distribution Algorithm (EDA) [39], a probabilistic technique that replaces the mutation and crossover operators, by sampling the probability distribution of the best individuals, to generate the new population, each generation. The probabilities start all equal and are updated each generation, based on the frequency of the chosen rules of the individuals with higher fitness. The proposed approach had a similar performance when compared with GE.

Later, Kim et al. [15] proposed Probabilistic Model Building Grammatical Evolution (PMBGE), which utilises a Conditional Dependency Tree (CDT) that represents the relationships between production rules used to calculate the new probabilities. Similar to [14], the EDA mechanism was implemented instead of the genetic operators. The results showed no statistical differences between GE and the proposed approach.

2.2.2 Improvements to GE

To address some of the main criticisms of GE, several improvements have been proposed in the literature related to the population initialisation [19, 20, 21, 22, 23], grammar design [5, 10, 11, 12, 40] and the representation of individuals [13, 14, 15, 16, 17, 18, 37].

The main method of population initialisation used in GE is random initialisation that consists of generating random genotypes with a fixed size and then perform the mapping. Although commonly used, this has been criticized for creating many invalid individuals [11].

One of the first improvements that emerged was the adaptation of ramped half-and-half [30] to GE, also referred to as sensible initialisation [20]. The adaptation was done using the GE grammar to choose the branches. The expansion is made from the leftmost leaf to the right. Despite having shown that it produces dense trees for some problems (some programs have asymmetric shapes, which is difficult to achieve with dense trees), this approach is the most recommended in the literature [34].

The Tree-Adjunct Grammatical Evolution (TAGE) [23] uses a Tree-Adjunct Grammar (TAG) [41] so that the language has an influence on the algorithm. A TAG is defined by a quintuple (Σ, NT, I, A, S) where Σ is a finite set of terminal symbols, NT is a finite set of non-terminal symbols with $\Sigma \cap NT = \emptyset$, S is the start symbol with $S \in NT$, I is the finite set of finite trees called initial trees and A is the finite set of finite trees called auxiliary trees. The initial and auxiliary trees interior nodes are labeled by non-terminal symbols, and the frontier nodes labeled by terminal and non-terminal symbols. The non-terminal symbols on the frontier of the trees are marked for substitution, except the foot node in

the auxiliary trees set. The individual's derivation tree is made up of several elementary trees. A better explanation of this algorithm can be found in the literature [23, 41]. This approach did not show significant differences in relation to GE, however in some problems it generated better solutions.

Position Independent Grow (PI Grow) [21] is an initialisation method inspired by πGE [13] mapping. This process uses a queue, in order to randomly choose the non-terminal symbols to be expanded. This approach showed to remove some bias that is present in sensible initialisation and that start with fewer invalid individuals [19, 21].

The Probabilistic Tree Creation 2 (PTC2) was initially proposed by Sean Luke [22]. This approach takes into account tree size rather than depth, which has proved to be an advantage in the creation of individuals [19]. Another advantage of these methods is that their computational complexity is relatively low. This algorithm keeps a list of the non-terminal symbols in the parse tree and randomly chooses which to expand until the tree size limit is reached. In [11] Harper compared different initialisation methods and showed that PTC2 achieved better performance in some problems, when comparing with Sensible Initialisation.

The choice of the grammar used in the problems showed to improve the performance, at the level of the search space used by the solutions [5, 42]. Because it has a big impact on the size of the solutions, some grammar design alternatives have been studied. Two methods that have shown significant improvements in the performance of GE are the use of recursively balanced grammars [5, 11] and the reduction of non-terminal symbols [5, 12].

The use of recursive grammars has been shown to generate many invalid individuals [11]. The recursively balanced grammars [5] add for each recursive production a non recursive production, in order to increase the probability of the non recursive productions being chosen. Although it has shown improvements over classic grammars, this results in more individuals consisting only of a non-terminal symbol [5].

An approach used to reduce the non-terminal symbols is to replace them by all their productions [12]. This method had a slight increase in performance, however it has the disadvantage of generating very complex and difficult to read grammars.

Grammatical Evolution by Grammatical Evolution ($(GE)^2$) [10] is an approach in which there is co-evolution of grammar and genetic code. The method uses two distinct grammars, the universal grammar and the solution grammar. The universal grammar dictates the structure of the solution grammar, that is used to map the individuals. This method has shown to be effective in evolving biases towards some non-terminal symbols. Later, was implemented into a new algorithm, meta-Grammar Genetic Algorithm (mGGA) [40], which obtained performance improvements.

2.2.3 Structured Grammatical Evolution

In this section we will present the SGE algorithm in detail, since it plays an essential role in the development of the proposed work. The Dynamic Structured Grammatical Evolution (DSGE) [6] was created to overcome some of the limitations presented by SGE [16]. Both versions were compared, and DSGE had equal or superior performance, statistically validated, in all the problems studied. Thus, from now on we will refer to DSGE by SGE.

SGE [6] is an alternative genotypic representation to GE, which tackles its limitations, resulting in higher locality and lower redundancy [16, 25]. In [37] different grammar-based GP approaches were compared, and the authors showed that SGE achieved a good

performance when compared with several grammar-based GP representations.

In SGE the genotype is a set of lists of ordered integers. Each list is associated to a non-terminal symbol and each element of the list represents the index of a derivation rule to be expanded to create the individual. An example of the genotype of an individual is depicted in Fig. 2.8 using the grammar presented in Fig. 2.7a. All the elements of the list are necessary to create the phenotype of the individual. In order to limit the size of the solutions, a maximum tree-depth value is previously defined from which only non-recursive productions can be chosen.

$< expr >$	$< op >$	$< var >$
[0, 1, 1]	[1]	[2, 0]

Figure 2.8: Example of a SGE genotype

The genotype is generated recursively, simulating the mapping into a phenotype until only terminal symbols remain. The pseudocode for creating the genotype for an individual is depicted in Algorithm 6. The algorithm receives as parameter the individual's genotype (which initially is empty), the non-terminal symbol to be expanded (which initially is the axiom of the grammar), the depth value of the symbol to be expanded (which begins at 0), the maximum depth limit of the trees defined, and the grammar to be used.

Algorithm 6 Random Candidate Solution Generation

```
1:  procedure CREATEINDIVIDUAL(genotype, symbol, current_depth, max_depth, grammar)
2:      if current_depth > max_depth then
3:          if is_recursive(symbol) then
4:              selected_rule = generate_terminal_expansion(symbol, grammar)
5:          end if
6:      else
7:          selected_rule = generate_expansion(symbol, grammar)
8:      end if
9:      if symbol in genotype then
10:         genotype[symbol].append(selected_rule)
11:     else
12:         genotype[symbol] = [selected_rule]
13:     end if
14:     expansion_symbols = grammar[symbol][selected_rule]
15:     for sym in expansion_symbols do
16:         if not is_terminal(sym) then
17:             createIndividual(genotype, sym, current_depth + 1, max_depth, grammar)
18:         end if
19:     end for
20: end procedure
```

The elements of the genotype (i.e., codons) are chosen randomly from among the available production rules of a certain non-terminal (Alg. 7). To avoid generating large solutions, only non-recursive productions can be chosen when the maximum depth limit is exceeded (Alg. 6 lines 2-5) using Alg. 8. After choosing the derivation rule, the codon (*selected_rule*) is added to the list of the respective non-terminal present in the genotype (Alg. 6 lines 9-13).

The mapping function (Alg. 9) is similar to the genotype initialisation function, and the productions are expanded according to the codons of the genotype. The mapping is

Algorithm 7 SGE function to select a expansion rule

```
1: procedure GENERATE_EXPANSION(symbol, grammar)
2:     selected_rule = randint(0, len(grammar[symbol] - 1))
3:     return selected_rule
4: end procedure
```

Algorithm 8 SGE function to select a non-recursive expansion

```
1: procedure GENERATE_TERMINAL_EXPANSION(symbol, grammar)
2:     non_rec = grammar.non_recursive(symbol)
3:     selected_rule = choice(non_rec)
4:     return selected_rule
5: end procedure
```

done by always expanding the leftmost non-terminal. The rule whose index is in the first position of the list associated with the non-terminal to be expanded is chosen. As in the initialisation, if the maximum depth limit has been reached, only non-recursive productions can be selected.

Table 2.1: SGE genotype-phenotype mapping example.

Derivation Step	Integers left
<expr>	[[0,1,1],[1],[2,0]]
<expr><op><expr>	[[1,1],[1],[2,0]]
<var><op><expr>	[[1],[1],[2,0]]
1.0<op><expr>	[[1],[1],[0]]
1.0-<expr>	[[1],[ ],[0]]
1.0-<var>	[[ ],[ ],[0]]
1.0-x	[[ ],[ ],[ ]]

Table 2.1 shows an example of the mapping, using the example genotype of Fig. 2.8 and the grammar of Fig. 2.7a. The mapping starts with the axiom of the grammar, in this case $< expr >$, and is expanded to the production rule of index 0, since this is the first codon in the list associated with the non-terminal $< expr >$. The index corresponds to the production rule $< expr >< op >< expr >$, so the next token to expand is $< expr >$. The next codon in the list is 1, it will be expanded to the respective rule, i.e., $< var >$. Because this is the leftmost non-terminal, to expand $< var >$ we have to take the first element of this non-terminal list, which is 2. Index 2 corresponds to derivation rule 1.0, and as this is a terminal symbol, it becomes definitive, moving on to the next non-terminal $< op >$. The process is repeated until there are no more non-terminals to expand.

In case genotypic changes have occurred, and there are no more available integers in the list of the non-terminal to be expanded, new codons are generated and added as needed (Alg. 9, lines 9-16).

The crossover method applied in SGE uses two parents to generate the offspring. A binary mask of the size of the number of lists of the genotype (the same number of non-terminals of the grammar) is randomly generated. The genotype of the offspring is created according to the values of the mask.

Fig. 2.9 shows an example of a crossover between two individuals (Fig. 2.9a), generating one offspring. Fig. 2.9b shows the randomly generated mask for the procedure. In the example shown, the descendant inherits the non-terminal $< expr >$ list from the second

Algorithm 9 SGE mapping function.

```
1: procedure MAPPING(genotype, positions_to_map, symbol, current_depth, max_depth, grammar)
2:     phenotype = ""
3:     if symbol not in positions_to_map then
4:         positions_to_map[symbol] = 0
5:     end if
6:     if symbol not in genotype then
7:         genotype[symbol] = [ ]
8:     end if
9:     if positions_to_map[symbol] >= len(genotype[symbol]) then
10:        if current_depth >= max_depth then
11:            selected_rule = generate_terminal_expansion(symbol, grammar)
12:        else
13:            selected_rule = generate_expansion(symbol, grammar)
14:        end if
15:        genotype[symbol].append(selected_rule)
16:    end if
17:    gen_int = genotype[symbol][positions_to_map[symbol]]
18:    expansion = grammar[symbol][gen_int]
19:    positions_to_map[symbol] += 1
20:    for sym in expansion do
21:        if is_terminal(sym) then
22:            phenotype += sym
23:        else
24:            phenotype += mapping(genotype, positions_to_map, sym, current_depth
   + 1, max_depth, grammar)
25:        end if
26:    end for
27:    return phenotype
28: end procedure
```

parent, and the remaining ones from the first. In case two descendants are generated, the other would get the opposite lists.

Parent 1:

$< expr >$	$< op >$	$< var >$
[0, 1, 1]	[1]	[2, 0]

Mask:

$< expr >$	$< op >$	$< var >$
1	0	0

Parent 2:

$< expr >$	$< op >$	$< var >$
[0, 0, 0, 1, 1, 1]	[2, 3]	[2, 0, 1]

(a) Genotype before crossover.

Offspring:

$< expr >$	$< op >$	$< var >$
[0, 0, 0, 1, 1, 1]	[1]	[2, 0]

(b) Genotype after crossover.

Figure 2.9: Example of SGE's crossover between two individuals, generating one offspring.

Unlike crossover, mutation occurs within genotype lists. Only lists that contain more than one expansion possibility are considered. The values to be mutated are chosen randomly, and are replaced by another valid possibility. Fig. 2.10 shows an example of a mutation in an individual. Considering that the first element of the list of the non-terminal $< op >$, and the second list element of the non-terminal $< var >$ have been randomly selected, these codons will be replaced by a valid option other than the original (Fig. 2.10b).

$< expr >$	$< op >$	$< var >$
[0, 1, 1]	[**1**]	[2, **0**]

(a) Genotype before mutation.

$< expr >$	$< op >$	$< var >$
[0, 1, 1]	[**3**]	[2, **1**]

(b) Genotype after mutation.

Figure 2.10: Example of SGE's mutation on two codons of the genotype.

2.3 Validation Metrics

To effectively evaluate an algorithm, different metrics can be used. In this work we will consider performance, locality, and redundancy as metrics to make a more complete analysis of the algorithms. The performance, allows us to compare algorithms based on the fitness assigned to individuals, giving an overview of their behaviour in a given problem. The locality and redundancy metrics analyse the representation of individuals and the genotype-phenotype mapping process along with the impact of genetic operators, to examine the effectiveness of the evolutionary search.

Within the literature, different approaches have already been used to study locality and redundancy, so the work done by Raidl et al. [26] and Lourenço et al. [16] will be followed. To understand how these two metrics will be analysed, it is necessary to introduce some concepts first. The phenotypic distance (i.e., the distance between two phenotypes), d^p, is calculated by recurring to the metric *edit distance*. This metric calculates the similarity between two trees, i.e., the minimum number of operations required to turn one tree into another. Different operations can be considered, such as insertion, deletion and replacement, which are applied to the tree nodes.

Since locality and redundancy reflect the impact of genetic operators on the mapping process, they are therefore studied based on the Mutation Innovation (MI) criteria [26]. The MI is calculated with the phenotypic distance before and after mutation occurs. This variable allows the analysis of the frequency with which a mutation causes alterations in the phenotype and its impact. Let x be a solution in the search space and x^m the solution

after applying mutation. Considering X and X^m the respective phenotypes, we can define MI as:

$$MI = d^p(X, X^m) \tag{2.1}$$

2.3.1 Redundancy

The redundancy corresponds to the existence of several genotypes that can represent the same phenotype. This is a phenomenon that can also be observed in nature through neutral mutations, however in excess it can decrease the exploration of the search space, reducing the efficiency of the algorithm. Past research shows that GE suffers from high redundancy [4, 25].

Based on the definition, redundancy can be studied by analysing the proportion of non-effective mutations (MI = 0).

2.3.2 Locality

Locality refers to how changes in genotype affect the phenotype. It is considered strong if small changes in genotype result in small changes in the phenotype, resulting in a better local search for solutions. It has been extensively studied in the literature that GE suffers from low locality [3, 24, 25].

The analysis of the locality can be done by calculating the phenotypic distance between a random individual and a individual resulting from a mutation that caused changes in phenotype (MI $>$ 0) [26].

2.3.3 Performance

The analysis of an algorithm can be evaluated with different metrics, depending on the intended analysis. These metrics usually relate to the quality of the solutions (i.e., effectiveness) or speed of the algorithm (i.e., efficiency) [28]. Due to the stochasticity of evolutionary algorithms, each experiment has to be repeated over several independent runs, keeping the same parameters, and then make a statistical analysis of the results to reject, or not, the null hypothesis. The type of statistical test performed depends on the data used.

Depending on the desired metric, different values can be collected. Following we will show some of the most commonly used measures. In the case of effectiveness, this can be studied with the success rate, i.e., the percentage of runs in which the success criterion is achieved is verified. This success criterion can be, for example, finding the solution to the problem or when the fitness exceeds a defined value. Another measure of effectiveness, is to analyse the best fitness ever at each generation of all runs, which can be useful especially when the main interest is to obtain the best possible result. The same analysis can be done with the worst fitness ever. The most used metric for comparing evolutionary algorithms is the mean best fitness, in which the best fitness of each generation of all runs is averaged.

The design of the fitness function is important for the evolution of solutions, it varies according to the problems, and can be designed with the objective of maximizing or minimizing the fitness of the solutions. Although it is possible to adapt the algorithms to

real world problems, being only necessary to design the fitness function and, in the case of grammar-based GP, the grammar, it is important to test the algorithms in known problems, with different degrees of difficulty to be able to evaluate their behaviour in different known environments. McDermott et al. [43], made a collection of the most used problems by the GP community and their fitness functions, which covers problems of symbolic regression, classification, model prediction and others.

This page is intentionally left blank.

Chapter 3

Probabilistic Grammatical Evolution

In this chapter two new probabilistic variants of Grammatical Evolution (GE) are proposed, namely Probabilistic Grammatical Evolution (PGE) and Co-evolutionary Probabilistic Grammatical Evolution (Co-PGE). The main distinction between these methods and the original GE, is concerned with the representation of individuals and the mapping method.

3.1 Representation and Mapping

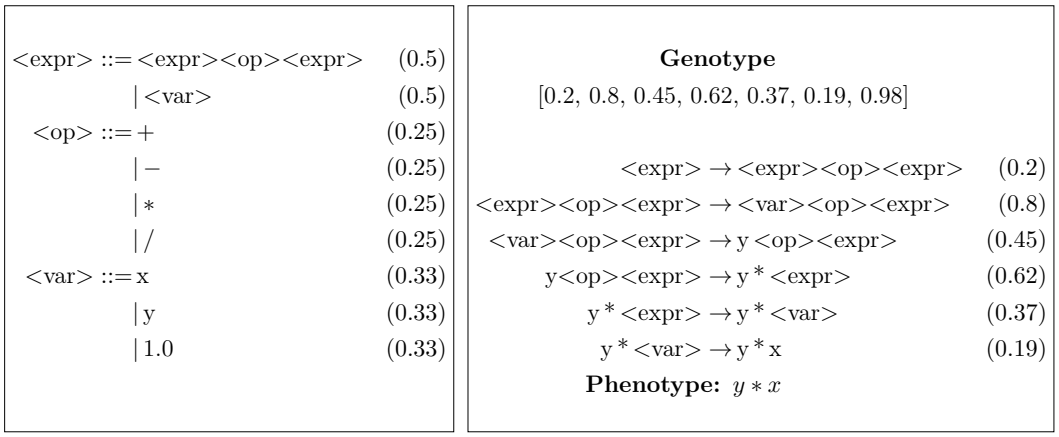
PGE is the first representation and mapping mechanism proposal for GE. In PGE the genotype-phenotype mapping is performed by using a Probabilistic Context-Free Grammar (PCFG) and the genotype of individuals is a list of real values, where each value represents the probability of choosing a certain derivation rule.

A PCFG is a quintuple $PG = (NT, T, S, P, Probs)$ where NT and T represent the non-empty set of *Non-Terminal (NT)* and *Terminal (T)* symbols, S is an element of NT called the axiom, P is the set of production rules, and $Probs$ is a set of probabilities associated with each production rule.

The overall mapping procedure is shown in Alg. 10. The algorithm takes as argument the genotype of the individual and the PCFG, and starts at the axiom of the grammar (Alg. 10, lines 2-3). One iteration is performed for each codon of the genotype (Alg. 10, line 4), which is used to choose the production rule to expand from the leftmost non-terminal. The algorithm goes through the productions of the non-terminal, and checks to which rule the codon corresponds (Alg. 10, lines 8-14). The process ends when the phenotype is valid (i.e., only terminal symbols) (Alg. 10, lines 16-17) or when there are no more codons in the genotype.

Fig. 3.1 shows an example of the application of the PGE mapping mechanism. The panel on the left (Fig. 3.1a) shows a PCFG, where each derivation rule has a probability associated. The set of non-terminals is composed of $< expr >$, $< op >$ and $< var >$. The right panel (Fig. 3.1b) shows how the mapping procedure works.

We start by taking the first value of the genotype, which is 0.2, and the axiom of the grammar, $< expr >$, which has two possible rules for expansion. In the beginning of the mapping process, all derivation options for each production rule have the same probability of being selected. In the case of the axiom, we have two derivation options, each one having a probability of 0.5 of being selected. Projecting the value of the codon on that



(a) Example of a PCFG. (b) Genotype-phenotype mapping.

Figure 3.1: Example of genotype-phenotype mapping with PCFG.

scale, it would be in the first range ($0.2 < 0.5$), which results in the first rule being chosen ($< expr >< op >< expr >$) to rewrite $< expr >$. The next symbol to expand is $< expr >$ because it is the leftmost non-terminal, and the codon to use is 0.8. As we have seen in the previous expansion, there are two options available. Since 0.8 is in the second range, the second production is chosen ($< var >$). This process is repeated until there are no more non-terminals left to expand, or no more codons left in the genotype. When this last situation occurs, we can use a wrapping mechanism similar to the standard GE, where the genotype will be reused a certain number of times. If after the wrapping we still have not mapped the individual completely, the mapping process stops, and the individual is considered invalid.

Algorithm 10 Mapping with PCFG

```
1: procedure GENERATEINDIVIDUAL(genotype, grammar)
2:     start = grammar.getStart()
3:     phenotype = [start]
4:     for codon in genotype do
5:         symbol = phenotype.getNextNT()
6:         productions = grammar.getRulesNT(symbol)
7:         cum_prob = 0.0                          ▷ Cumulative Sum of Probabilities
8:         for prod in productions do
9:             cum_prob = cum_prob + prod.getProb()
10:            if codon ≤ cum_prob then
11:                selected_rule = prod
12:                break
13:            end if
14:        end for
15:        phenotype.replace(symbol, selected_rule)
16:        if phenotype.isValid() then
17:            break
18:        end if
19:    end for
20: end procedure
```

Initially, all productions of the PCFG have the same probability of being chosen, however, each generation, after evaluating the population, the probabilities are adjusted according

to the productions used to generate the best individual of the current generation or the best individual overall. Alternating between these two bests helps us to avoid using the same individual in consecutive generations to adjust the probabilities of the PCFG, balancing global exploration with local exploitation. Additionally, we use a learning factor to make the transitions on the search space smoother.

To update the probabilities in the grammar, we use Alg. 11, where j is the number of productions of a non-terminal symbol of the grammar, i is the index of the production which probability is being updated and λ is the learning factor. All probabilities of the productions of each non-terminal symbol are updated. If the production was used to create the individual, the probability is increased (Alg. 11, line 5) else, if a production is not present in the individual, then this value will be reduced (Alg. 11 line 7). The *min* operator ensure that when we update the probabilities they are not greater than 1. After updating the probabilities of the PCFG, it is used to update the phenotype of the individuals for the next generation.

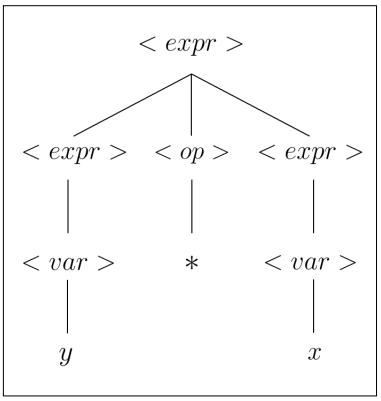
Algorithm 11 Probabilistic Grammatical Evolution

1: **procedure** UPDATEPROBABILITIES($individual, grammar$)
2: $counter = individual.getCounter()$
3: **for** NT, productions **in** grammar **do**
4: **for each** production rule i **of** prods **do**
5: **if** $counter_i > 0$ **then**
6: $prob_i = min(prob_i + \lambda * \frac{counter_i}{\sum_{k=1}^{j} counter_k}, 1.0)$
7: **else**
8: $prob_i = prob_i - \lambda * prob_i$
9: **end if**
10: **end for**
11: **while** $\sum_{k=1}^{j} prob_i \neq 1.0$ **do**
12: extra = (1.0 - $\sum_{k=1}^{j} prob_i$) / j
13: **for each** production rule i of prods **do**
14: $prob_i = prob_i + extra$
15: **end for**
16: **end while**
17: **end for**
18: **end procedure**

After the update of the probabilities for each derivation rule, we make sure that the sum of the probabilities of all derivation rules, for each non-terminal, is 1. If the sum surpasses this value, the excess is proportionally subtracted from the derivation options for a non-terminal. If the sum is smaller than one, the missing amount is added equally to the production rules of the non-terminal.

We are going to use the example presented in Fig. 3.1, to show how the probabilities are updated. On the left panel of Fig. 3.2 we can see the derivation tree of the individual used (Fig. 3.2a) to update the probabilities of the PCFG on the right panel (Fig. 3.2b). The learning factor used was $\lambda = 0.01$ (1%).

Looking at the derivation tree (Fig. 3.2a), we can see that the first derivation rule ($<expr><op><expr>$) of the non-terminal $<expr>$ was expanded once, and the second derivation rule ($<var>$) was expanded twice. Using the Alg. 11, the new probability for the first derivation option is, approximately, 0.503(3) ($min(0.5 + 0.01 * \frac{1}{3}, 1)$), and for the second rule, approximately, 0.506(6) ($min(0.5 + 0.01 * \frac{2}{3}, 1)$). As the sum of the



(a) Derivation tree of the individual used to update the PCFG.

New Grammar

<expr> ::=	<expr><op><expr>	(0.498)
	| <var>	(0.502)
<op> ::=	+	(0.247)
	| −	(0.247)
	| ∗	(0.259)
	| /	(0.247)
<var> ::=	x	(0.336)
	| y	(0.336)
	| 1.0	(0.328)

(b) Updated grammar.

Figure 3.2: Example of grammar update in PGE.

probabilities surpass 1, the excess $((0.506(6) + 0.503(3) - 1)/2 = 0.00495)$ is subtracted from both probabilities and the value is rounded to 3 decimal places. Then we distribute the excess for the derivations rules: the first rule is updated to 0.498 and the second rule is updated to 0.502.

This process is applied to the other symbols. For the non-terminal $< op >$ the rules $+$, $-$, and / were never chosen so they will update equally to 0.2475 $(0.25 - 0.01 * 0.25)$, and the multiplication operator, $*$, was selected once, updating to 0.26 $(min(0.25 + 0.01 * \frac{1}{1}, 1))$. As the sum of the four probabilities surpasses one, the excess (0.0025) must be divided by four and subtracted equally. The result is rounded, staying with 0.247 for the $+$, $-$, and / symbols, and 0.259 for the $*$ symbol.

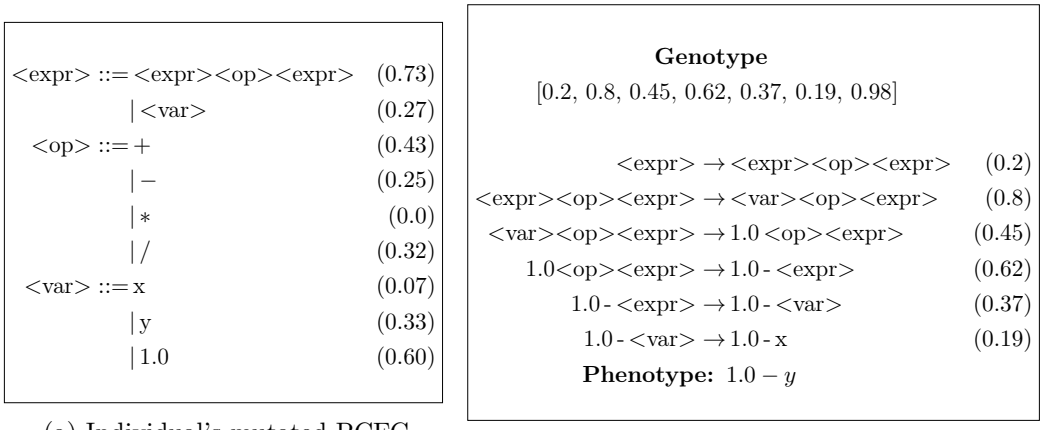
The non-terminal $< var >$ was expanded once for the terminal x and y, and never expanded for the terminal 1.0. By applying the algorithm, the rules x and y should be updated to 0.335 $(min(0.33 + 0.01 * \frac{1}{2}, 1))$ and the terminal 1.0 to 0.3267 $(0.33 - 0.01 * 0.33, 0)$, being the sum of the probabilities different than one, the adjustment and rounding should be done and they are updated to 0.336 and 0.328, respectively.

After the population is subjected to the genetic operators and the PCFG is updated, the population is remapped using the new updated grammar.

3.2 Co-evolutionary Approach

Co-PGE is a new variant of GE that uses the same genotype representation and mapping method introduced by PGE. The genotype of the individuals is an array of floats, and the mapping is done using a Probabilistic Structured Grammatical Evolution (PSGE). The difference between the methods is in the way the probabilities of the PCFG are updated. In Co-PGE, each individual has an associated PCFG and both the genotype and the PCFG co-evolve as they are subjected to selective pressure. An example of an individual and its PCFG is presented in Fig. 3.3.

In the beginning of the evolutionary process, the genotype, which is a list of floats, is randomly generated, and to each individual is assigned a PCFG whose probabilities are equal for all productions rules of each non-terminal. Then, at each generation, individuals are subjected to selection and variation operators, as well as the associated grammar.



(a) Individual's mutated PCFG.

(b) Genotype-phenotype mapping.

Figure 3.3: Example of Co-PGE individual.

Alg. 12 shows the pseudo-code of the mutation process of the PCFG in an individual. The algorithm takes as parameters the individual whose grammar is to be mutated, the probability of mutation occurring (between 0 and 1) and the Gaussian standard deviation to use. For each production rule of each non-terminal is generated a random value to verify if the mutation should occur (Alg. 12, line 5). At most, only one production rule of each non-terminal can be mutated, per generation. For each production rule selected to mutate, is added a value generated with a Gaussian distribution to its probability (Alg. 12, lines 6-7). Gaussian mutations have been widely used in the literature and have showed to be a good approach to make small changes in the search space [44, 45, 46]. The values of the other productions of the same non-terminal are changed according to the new probability, until the sum equals one (Alg. 12, line 9). During crossover, the offspring inherits the grammar of the parent with the best fitness. At the end of each generation, the phenotype of the individual is updated using the new grammar and genotype, following the mapping process presented for PGE in Alg. 10.

Algorithm 12 Co-evolutionary Probabilistic Grammatical Evolution

```
1: procedure MUTATEGRAMMAR(individual, prob_mutation_grammar, sd_normal_distribution)
2:     grammar = individual.getGrammar()
3:     for NT, prods in grammar do
4:         for each production rule i of prods do
5:             if random() < prob_mutation_grammar then
6:                 gauss_value = random_Gaussian(0, sd_normal_distribution)
7:                 prob_i = min(prob_i + gauss_value, 1.0)
8:                 prob_i = max(prob_i, 0.0)
9:                 adjust_probabilities(prods)
10:                break
11:            end if
12:        end for
13:    end for
14: end procedure
```

Fig. 3.4 shows an example of a grammar before (Fig. 3.4a) and after (Fig. 3.4b) suffering a mutation. Assuming that the non-terminal $< expr >$ production rule $< var >$ has been randomly selected to be mutated, and that the random number generator of a Gaussian

<expr> ::= <expr><op><expr>	(0.5)
| <var>	**(0.5)**
<op> ::= +	(0.25)
| −	(0.25)
| ∗	(0.25)
| /	(0.25)
<var> ::= x	**(0.33)**
| y	(0.33)
| 1.0	(0.33)

(a) PCFG grammar before mutation.

<expr> ::= <expr><op><expr>	(0.73)
| <var>	**(0.27)**
<op> ::= +	(0.25)
| −	(0.25)
| ∗	(0.25)
| /	(0.25)
<var> ::= x	**(0.45)**
| y	(0.27)
| 1.0	(0.27)

(b) PCFG grammar after mutation.

Figure 3.4: Example of grammar mutation in Co-PGE

distribution of mean 0 and standard deviation 0.50 ($N(0, 0.50)$) has generated the number -0.23, the new probability of that production becomes 0.27 ($0.50 - 0.23$). The probability of the other production of the non-terminal $< expr >$ is adjusted to 0.73. As a maximum of one mutation can occur in one production of each non-terminal, other mutations can occur in other non-terminals, for example, if the production rule x of the non-terminal $< var >$ suffers a mutation of $+0.12$, it will have a value of 0.45, and the remaining outputs need to be adjusted, having a probability of 0.27.

This page is intentionally left blank.

Chapter 4

Probabilistic Structured Grammatical Evolution

Mainly due to its flexibility, Grammatical Evolution (GE) has been widely used by researchers and practitioners from various research fields, obtaining good results in multiple applications. However it is not exempt from some known issues, such as low locality and high redundancy. These issues have been targeted by several works in the literature, and in recent years some proposals have emerged that addresses them. One method that has shown to be able to overcome these problems and outperform is Structured Grammatical Evolution (SGE).

Given the benefits in terms of performance, the methods proposed in the previous chapters were adapted to SGE, creating two new variants that introduce a new representation and mapping method.

4.1 Representation and Mapping

Probabilistic Structured Grammatical Evolution (PSGE) is an hybrid approach between the representation and mapping mechanism used by SGE and Probabilistic Grammatical Evolution (PGE). The process of generating the individuals and the mapping are similar to that of SGE, but the representation of the individuals is different, with the codons being probabilities rather than integers. The genotype-phenotype mapping relies on a Probabilistic Context-Free Grammar (PCFG), and the probabilities of the grammar are updated every generation, using the mechanism proposed for PGE.

Similarly to SGE, the genotype of the individuals is a set of dynamic lists, each being associated with a non-terminal of the grammar. Each list has an ordered sequence of real values, that correspond to the probability of selecting production rules. The pseudo-code of the initialization of individuals in PSGE is presented in Alg. 13.

The initialization of individuals is done recursively, and at each iteration a codon is randomly generated (Alg. 13, line 2) and added to the genotype list of the non-terminal that is being expanded (Alg. 13, lines 10-14). The algorithm takes as arguments the genotype (which starts empty), the non-terminal symbol to expand (in the first iteration it is the axiom of the grammar), the current depth (starts at 0), the maximum depth limit, and the grammar. The function simulates the mapping process (Alg. 13, lines 3-9), to know which are the next non-terminals to expand. The algorithm ends when all the non-terminals

symbols have been expanded until only terminals remain, i.e., the genotype corresponds to a valid individual.

Algorithm 13 Random Candidate Solution Generation of PSGE

```
1: procedure CREATEINDIVIDUAL(genotype, symbol, current_depth, max_depth, grammar)
2:     codon = random()
3:     if current_depth > max_depth then
4:         if is_recursive(symbol) then
5:             selected_rule = generate_terminal_expansion(symbol, codon, grammar)
6:         end if
7:     else
8:         selected_rule = generate_expansion(symbol, codon, grammar)
9:     end if
10:    if symbol in genotype then
11:        genotype[symbol].append(codon)
12:    else
13:        genotype[symbol] = [codon]
14:    end if
15:    expansion_symbols = grammar[symbol][selected_rule]
16:    for sym in expansion_symbols do
17:        if not is_terminal(sym) then
18:            createIndividual(genotype, sym, current_depth + 1, max_depth, grammar)
19:        end if
20:    end for
21: end procedure
```

The pseudo-code of the genotype-phenotype mapping is presented in Alg. 14. The algorithm receives as argument the genotype, a counter called *positions_to_map* (initialised empty), the symbol to expand (starts in the axiom), the current depth, the maximum depth limit and the grammar. The *positions_to_map* counter is used to store the genotype position of the next codon to be used to expand each non-terminal. If, during mapping, more codons are needed to create a valid individual, they will be randomly generated and added to the genotype (Alg. 14, lines 9-12). The dynamic genotype is one of the advantages of the representation proposed by SGE. With the depth limit, it is possible to add productions whenever necessary, without the risk of bloat (a considerable growth in the size of the solutions [28]), always creating valid individuals.

The process of selecting an expansion rule from a real value with a PCFG (Alg. 14, line 17) is similar to the one used in PGE, however, if the defined maximum tree depth limit is exceeded, only non-recursive productions will be selected (Alg. 14, lines 14-16). The function to choose the production rule to expand is depicted in Alg. 15. The function receives as parameter the non-terminal symbol to be expanded, the codon, and the grammar. It is verified whether the codon belongs to the probability range of each production rule of the non-terminal to be expanded. When this condition is verified, the rule is chosen. When the defined maximum tree depth limit is exceeded we use the function presented in Alg. 16. The function receives as parameter the non-terminal symbol to be expanded, the codon, and the grammar. The algorithm considers only non-recursive rules (Alg. 16, line 3), and adjusts the probabilities of each of them, so that the sum is 1. To accomplish this, we first sum the value of the current probabilities of the non-recursive rules (Alg. 16, line 3), which is used to perform the adjustment (Alg. 16, line 6). Using the new probabilities, the production rule is chosen with the normal procedure: the production rule whose interval the codon lies in is chosen.

Algorithm 14 Genotype-Phenotype Mapping Function of PSGE

```
1: procedure MAPPING(genotype, positions_to_map, symbol, depth, max_depth, grammar)
2:     phenotype = ""
3:     if symbol not in positions_to_map then
4:         positions_to_map[symbol] = 0
5:     end if
6:     if symbol not in genotype then
7:         genotype[symbol] = [ ]
8:     end if
9:     if positions_to_map[symbol] >= len(genotype[symbol]) then
10:        codon = random()
11:        genotype[symbol].append(codon)
12:    end if
13:    codon = genotype[symbol][positions_to_map[symbol]]
14:    if depth >= max_depth then
15:        selected_rule = generate_terminal_expansion(symbol, codon, grammar)
16:    else
17:        selected_rule = generate_expansion(symbol, codon, grammar)
18:    end if
19:    expansion = grammar[symbol][selected_rule]
20:    positions_to_map[symbol] += 1
21:    for sym in expansion do
22:        if is_terminal(sym) then
23:            phenotype += sym
24:        else
25:            phenotype += mapping(genotype, positions_to_map, sym, depth + 1,
    max_depth, grammar)
26:        end if
27:    end for
28:    return phenotype
29: end procedure
```

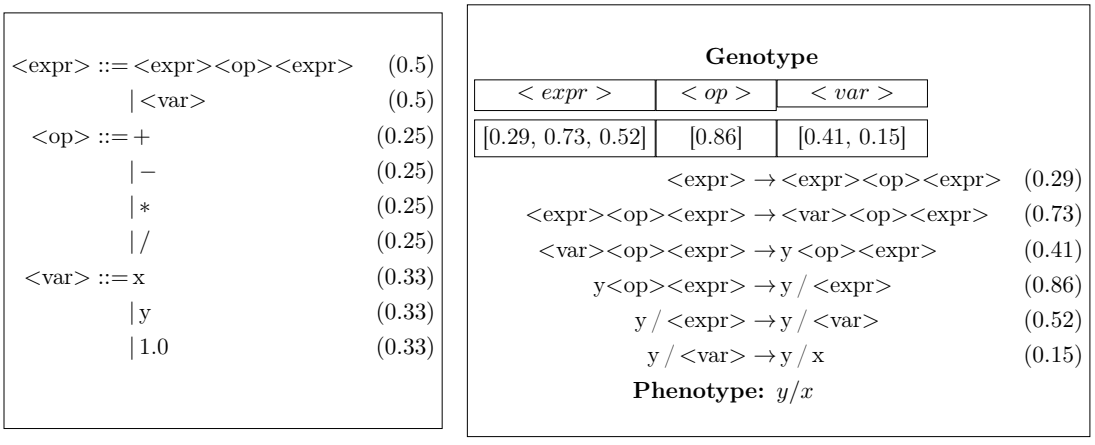
Algorithm 15 PSGE function to select a expansion rule

```
1: procedure GENERATE_EXPANSION(symbol, codon, pcfg)
2:     cum_prob = 0.0                          ▷ Cumulative Sum of Probabilities
3:     productions = pcfg[symbol]
4:     for prod in productions do
5:         cum_prob = cum_prob + prod.getProb()
6:         if codon ≤ cum_prob then
7:             selected_rule = prod
8:             break
9:         end if
10:    end for
11:    return selected_rule
12: end procedure
```

Algorithm 16 PSGE function to select a non-recursive expansion

```
1: procedure GENERATE_TERMINAL_EXPANSION(symbol, codon, pcfg)
2:     cum_prob = 0.0                                  ▷ Cumulative Sum of Probabilities
3:     non_recursive_prods = get_non_recursive_prods(pcfg[symbol])
4:     total_non_recursive_prods = sum(terminal_prods.getProd())   ▷ Sum of
   probabilities of Non Recursive prods
5:     for prod in productions do
6:         new_prob = prod.getProb()/total_terminal_prods
7:         cum_prob = cum_prob + new_prob
8:         if codon ≤ cum_prob then
9:             selected_rule = prod
10:            break
11:        end if
12:    end for
13:    return selected_rule
14: end procedure
```

Fig. 4.1 shows an example of the mapping process in PSGE. The left panel (Fig. 4.1a) shows the PCFG used, and the right panel (Fig. 4.1b) shows the genotype of the individual and the mapping procedure. The mapping starts with the axiom of the grammar, $< expr >$, and at the first element of its respective list in the genotype, 0.29. The value of the codon is compared with the probabilities of the rules of the non-terminal. In this case we have two options, with equal probability. As 0.29 falls within the range of probabilities of the first production, the non-terminal will be expanded to $< expr >< op >< exp >$. Following the leftmost derivation rule, the next non-terminal to expand is $< expr >$. We repeat the process and this time the codon, 0.73, corresponds to the second derivation rule, $< var >$. The next non-terminal to expand is $< var >$, which has three derivation rules, and, since it has not yet been expanded, we take the first codon in the $< var >$'s list, which is 0.41. Looking at the probabilities of the derivation rules, we see that the codon falls within the range of the second rule, y. The process is repeated until a valid individual is formed.



(a) PCFG example.

(b) Genotype-phenotype mapping procedure.

Figure 4.1: Example of the genotype-phenotype mapping of PSGE with a PCFG.

At the end of each generation the best overall individual and the best individual from the current generation are used alternately to update the PCFG probabilities, using the same mechanism proposed for PGE.

For each production i of each non-terminal j we have a *counter* with the number of times that each production was chosen, and the probability (*prob*) of the PCFG for that production. If the counter is greater than zero, that is, the production rule was used to map the individual, we use Equation 4.1. If the counter is zero, that is, the production rule has not been used by the individual, we use Equation 4.2.

$$prob_i = min(prob_i + \lambda * \frac{counter_i}{\sum_{k=1}^{j} counter_k}, 1.0) \tag{4.1}$$

$$prob_i = prob_i - \lambda * prob_i \tag{4.2}$$

where λ is the learning factor, with $\lambda \in [0, 1]$.

After updating the probabilities using the equations, these are adjusted until the sum of the probabilities of each production rule of each non-terminal is 1.

4.1.1 Variation Operators

Individuals can be modified using genetic operators. Mutation occurs at the level of probability, that is, a new codon is randomly generated, which is later used to choose the new expansion rule. A Gaussian mutation [44, 45, 46] is applied in the chosen values, having this new value to be contained in the interval [0,1].

Fig. 4.2 shows an example of Gaussian mutation in an individual. Assuming that randomly only the first codon in the list of the non-terminal $< var >$ was selected for mutation and that the value generated with a Gaussian distribution of mean 0 and standard deviation 0.50 ($N(0, 0.50)$) was 0.23, the codon will be changed to 0.64 ($0.41 + 0.23$).

$< expr >$	$< op >$	$< var >$
[0.29, 0.73, 0.52]	[0.86]	[**0.41**, 0.15]

(a) Genotype before mutation.

$< expr >$	$< op >$	$< var >$
[0.29, 0.73, 0.52]	[0.86]	[**0.64**, 0.15]

(b) Genotype after mutation.

Figure 4.2: Example of PSGE's mutation on one codon of the genotype.

The crossover is the same as the SGE's. After being submitted to the genetic operators and updating the grammar, the individuals are remapped before passing to the next generation.

4.2 Co-evolutionary Approach

Co-evolutionary Probabilistic Structured Grammatical Evolution (Co-PSGE) is a method that uses the same representation and mapping introduced in the previous section, but each individual co-evolves a PCFG.

At initialisation, a genotype is generated for each individual as well as a PCFG. The genotype is a list of ordered lists, with one list for each non-terminal, and each element of the list is a real value. The genotype is initialized in the same way as in PSGE, and the pseudocode is present in Algorithm 13. The grammar assigned to the individual at initialization has the same probability for each production rule of each non-terminal. At each generation, individuals are subjected to selection and variation operators as well as the associated grammar of each individual.

The mutation performed in the PCFG is the same as proposed for Co-evolutionary Probabilistic Grammatical Evolution (Co-PGE), which occurs in the probabilities, with a maximum of one mutation per non-terminal symbol. The probabilities of the production rules selected for mutation are modified adding a random value generated from a Gaussian distribution. The probabilities of the remainder productions of each non-terminal that suffered mutation must be adjusted until the sum of the probabilities is 1. After updating the grammar, the individual's phenotype is remapped.

During crossover between two individuals, the offspring inherits the grammar of the parent with better fitness.

This page is intentionally left blank.

Chapter 5

Validation

The validation of the proposed methods will be done based on three different metrics: performance, redundancy, and locality. These metrics allow fair comparison between the proposed methods, Grammatical Evolution (GE) [1] and Structured Grammatical Evolution (SGE) [6].

5.1 Performance Analysis

The performance of different algorithms will be conducted by examining the evolution of the mean fitness of the solutions in different problems. For this, we rely on the framework proposed by Whigham et al. [9], in which six problems of different scopes are considered.

Table 5.1: Parameters used in the experimental analysis for GE, PGE, Co-PGE, SGE, PSGE, and Co-PSGE.

Parameters	GE	PGE	Co-PGE	SGE	PSGE	Co-PSGE
Population Size				1000		
Generations				50		
Elitism Count				100		
Mutation Rate				0.05		
Crossover Rate				0.90		
Tournament				3		
Size of Genotype		128			-	
Max Depth		-			10	

Table 5.1 presents the experimental settings used by all the approaches. Regarding the genetic operators used by SGE, Probabilistic Structured Grammatical Evolution (PSGE) and Co-evolutionary Probabilistic Structured Grammatical Evolution (Co-PSGE), these methods all use the same crossover, which is the one presented in Section 2.1.3. Regarding the mutation operator, in the case of SGE, the mutated codon is replaced with a different valid option, while in PSGE and Co-PSGE, a Gaussian mutation with $N(0, 0.50)$ in the codon value is performed. In what concerns the variation operators of GE, Probabilistic Grammatical Evolution (PGE) and Co-evolutionary Probabilistic Grammatical Evolution (Co-PGE), they use a one point crossover. The mutation of GE replaces the selected codons by a new one generated in the interval $[0, 255]$ and in the case of PGE and Co-PGE a float mutation is used, in which the codons are replaced by new ones generated in the interval $[0, 1]$. The wrapping mechanism was removed from GE, PGE and Co-PGE.

Additionally, PGE and PSGE use a learning factor of $\lambda = 1.0\%$, and, Co-PGE and Co-PSGE use a probability of occurring a mutation in the grammar of 5%, with a random value drawn from a $N(0, 0.50)$.

The fitness functions used to evaluate the individuals were designed with the objective of minimizing the error. In the case of Symbolic Regression and classification problems, the fitness is the Root Relative Squared Error (RRSE) (Equation 5.1) between the individual's solution (P) and the target (T) on a data set, with n being the number of samples. For the Boolean functions, the error is the number of incorrect predictions, and for the Path finding problem, the fitness is the number of pieces remaining after exceeding the step limit. All the problems are detailed bellow as well as the grammars used.

$$RRSE = \sqrt[2]{\frac{\sum_{i=1}^{n}(P_i - T_i)^2}{\sum_{i=1}^{n}(P_i - \bar{T})^2}}, \text{ with } \bar{T} = \frac{1}{n}\sum_{i=1}^{n} T_i \tag{5.1}$$

The results are the mean best fitness of each generation over 100 independent runs.

5.1.1 Symbolic Regression

Popular benchmark problem for testing Genetic Programming (GP) algorithms, with the objective of finding the mathematical expression that best fits a given dataset, using Grammar 5.1.

⟨*start*⟩ ::= ⟨*expr*⟩

⟨*expr*⟩ ::= ⟨*expr*⟩ ⟨*op*⟩ ⟨*expr*⟩ | (⟨*expr*⟩ ⟨*op*⟩ ⟨*expr*⟩)
| ⟨*pre_op*⟩ (⟨*expr*⟩) | ⟨*var*⟩

⟨*op*⟩ ::= + | - | * | /

⟨*pre_op*⟩ ::= sin | cos | exp | log | inv

⟨*var*⟩ ::= x[..] | 1.0

Grammar 5.1: Symbolic Regression grammar.

where $inv = \frac{1}{f(x)}$. The division and logarithm functions are protected, i.e., $1/0 = 1$ and $log(f(x)) = 0\ if f(x) \leq 0$.

Quartic polynomial

The goal of this problem is to find the expression defined by:

$$x^4 + x^3 + x^2 + x. \tag{5.2}$$

The function is sampled in the interval [-1, 1] with a step of 0.1.

Pagie polynomial

The aim is to find the following mathematical expression:

$$\frac{1}{1+x[1]^{-4}} + \frac{1}{1+x[2]^{-4}}. \tag{5.3}$$

The function is sampled with x[1], x[2] ∈ [-5, 5] with a step of 0.4.

5.1.2 Boston Housing Problem

This is a famous Machine Learning problem to predict the prices of Boston Houses. The dataset comes from the StatLib Library [47] and has 506 entries, with 13 features. It was divided in 90% for training and 10% for testing. The grammar used for the Boston Housing regression problem is Grammar 5.2.

⟨*start*⟩ ::= ⟨*expr*⟩

⟨*expr*⟩ ::= ⟨*expr*⟩ ⟨*op*⟩ ⟨*expr*⟩ | (⟨*expr*⟩ ⟨*op*⟩ ⟨*expr*⟩)
| ⟨*pre_ op*⟩ (⟨*expr*⟩) | ⟨*var*⟩

⟨*op*⟩ ::= + | - | * | /

⟨*pre_ op*⟩ ::= sin | cos | exp | log | inv

⟨*var*⟩ ::= x[1] |...| x[13] | 1.0

Grammar 5.2: Boston Housing grammar.

5.1.3 5-bit Even Parity

The objective of this problem is to evolve a boolean function that takes as input a binary string with length 5, and returns 0 if the string is even or 1 if it is odd. Considering $b0$, $b1$, $b2$, $b3$, and $b4$ the input bits, the grammar used is presented in Grammar 5.3.

⟨*start*⟩ ::= ⟨*B*⟩

⟨*B*⟩ ::= ⟨*B*⟩ and ⟨*B*⟩ | ⟨*B*⟩ or ⟨*B*⟩
| not (⟨*B*⟩ and ⟨*B*⟩) | not (⟨*B*⟩ or ⟨*B*⟩)
| ⟨*var*⟩

⟨*var*⟩ ::= b0 | b1 | b2 | b3 | b4

Grammar 5.3: 5-bit Even Parity grammar.

5.1.4 11-bit Boolean Multiplexer

The aim of the 11-bit Multiplexer is to decode a 3-bit binary address and return the value of the corresponding data register ($d0$ to $d7$). The function receives as input three addresses ($s0$ to $s2$) and eight data registers ($i0$ to $i7$). For this problem we used Grammar 5.4.

$\langle start \rangle$ ::= $\langle B \rangle$

$\langle B \rangle$ ::= $\langle B \rangle$ and $\langle B \rangle$ | $\langle B \rangle$ or $\langle B \rangle$ | not $\langle B \rangle$
| $\langle B \rangle$ if $\langle B \rangle$ else $\langle B \rangle$ | $\langle var \rangle$

$\langle var \rangle$ ::= s0 | s1 | s2 | i0 | i1 | i3 | i4| i5 | i6 | i7

Grammar 5.4: 11-bit Boolean Multiplexer grammar.

5.1.5 Santa Fe Artificial Ant

The Santa Fe Artificial Ant Problem is a path finding problem, whose goal is to evolve a program to guide an artificial ant in an environment, so it can collect food. The Santa Fe trail contains 89 food pellets distributed in a 32x32 grid and is depicted in Fig. 5.1. The agent starts at the top left corner, and can execute at most 650 steps. The set of rules to follow is defined by Grammar 5.5.

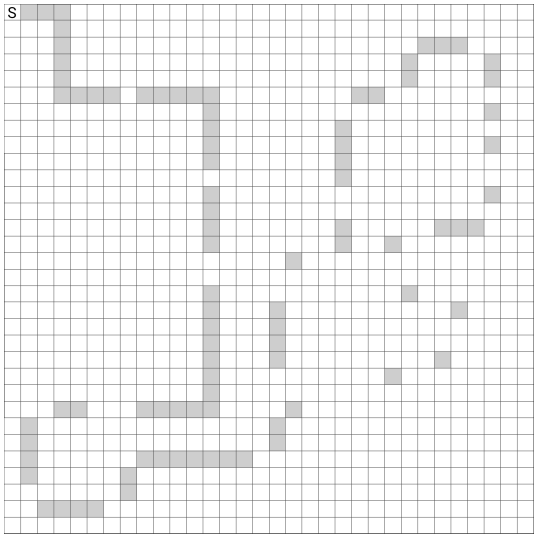


Figure 5.1: The Santa Fe trail.

5.1.6 Results

To better understand and compare the performance of all approaches we performed a statistical analysis. Since the results did not follow any distribution, and the populations were independently initialised, we employed the Kruskal-Wallis non-parametric test to check if there were meaningful differences between the different groups, i.e., approaches. When this happened we used the Mann-Whitney *post-hoc* test with Bonferroni correction to ascertain in which pairs this difference existed. For all the statistical tests we considered a significance level of $\alpha = 0.05$. The statistical tests were made between the proposed

⟨*start*⟩ ::= ⟨*code*⟩

⟨*code*⟩ ::= ⟨*line*⟩
| ⟨*code*⟩
⟨*line*⟩

⟨*line*⟩ ::= if ant.sense_food():
⟨*line*⟩
else:
⟨*line*⟩
| ⟨*op*⟩

⟨*op*⟩ ::= ant.turn_left() | ant.turn_right() | ant.move_forward()

Grammar 5.5: Santa Fe Artificial Ant grammar.

methods (PGE, Co-PGE, PSGE, and Co-PSGE), and the standard methods from the literature GE, shown in Table 5.2 and SGE, shown in Table 5.3.

Table 5.2: Results of the Mann-Whitney post-hoc statistical tests between the proposed methods and GE. Bonferroni correction is used and the significance level $\alpha = 0.05$ is considered. Values in bold mean that the proposed method is statistically better than GE.

p-value	PGE	Co-PGE	PSGE	Co-PSGE
Quartic Polynomial	**0.017**	0.101	**0.000**	**0.000**
Pagie Polynomial	**0.000**	**0.023**	**0.017**	**0.002**
Boston Housing Train	**0.000**	**0.000**	**0.000**	**0.000**
Boston Housing Test	**0.000**	**0.000**	**0.000**	**0.000**
5-bit Even Parity	**0.000**	**0.000**	**0.000**	**0.000**
11-bit Multiplexer	0.222	**0.046**	**0.000**	**0.000**
Santa Fe Ant Trail	0.567	0.052	**0.000**	**0.000**

Table 5.3: Results of the Mann-Whitney post-hoc statistical tests between the proposed methods and SGE. Bonferroni correction is used and the significance level $\alpha = 0.05$ is considered. Values in bold mean that the proposed method is statistically better than SGE.

p-value	PGE	Co-PGE	PSGE	Co-PSGE
Quartic Polynomial	0.000	0.000	0.149	**0.013**
Pagie Polynomial	0.290	0.007	0.006	0.077
Boston Housing Train	0.001	0.301	0.000	0.111
Boston Housing Test	0.030	0.256	0.288	**0.023**
5-bit Even Parity	0.000	0.000	0.066	**0.000**
11-bit Multiplexer	0.000	0.000	0.000	0.000
Santa Fe Ant Trail	0.000	0.000	0.158	0.044

Focusing on the statistical results of Tables 5.2 and 5.3, several observations about the approaches proposed in this work can be drawn. In bold are the p-values of the comparisons where the proposed methods are statistically better than GE (Table 5.2) and SGE (Table 5.3). Comparing the methods with GE (Table 5.2), we see that PGE and Co-PGE

are always better or equivalent, and that PSGE and Co-PSGE are always better in the problems considered. When comparing the approaches to SGE (Table 5.3), we can observe that PGE and Co-PGE have more statistical differences, making them worse, although they perform equally well in some problems. When we look at the PSGE column, we can see that it has never outperformed SGE, being statistically similar in four of the six problems; nevertheless, Co-PSGE has statistical differences and is better in three problems.

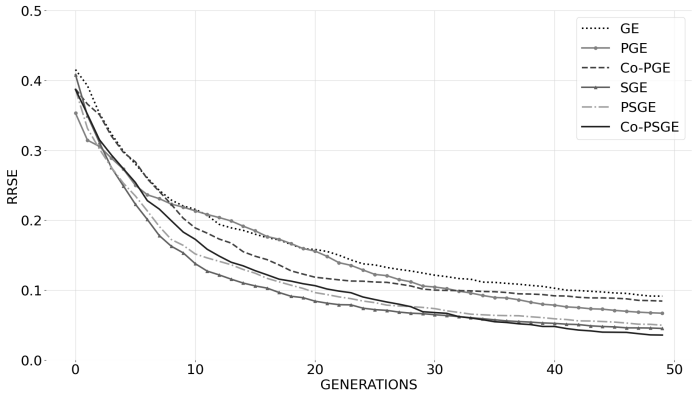


Figure 5.2: Performance results for the Quartic Polynomial. Results are the mean best fitness of 100 runs.

Fig. 5.2 shows the evolution of the mean best fitness for the quartic polynomial. Looking first at PGE and Co-PGE, we notice that both have a different behaviour. PGE starts with better fitness, however, around the 8th generation, it is overtaken by Co-PGE, and it has a much slower growth compared to the other methods. Around generation 30, PGE manages to recover and ends with better fitness than GE and Co-PGE. Co-PGE fitness decreases much faster, but stagnates around generation 20, and by the end of the evolutionary process it is caught by GE. This can be an indication that Co-PGE is converging faster, and that we might need to adjust the probabilities update mechanism. Looking at the statistical results between the methods and GE (Table 5.2), we can say that PGE is better than GE in this problem, presenting statistical differences. The Co-PGE presents a p-value greater than 0.05, which is an indication that the methods are statistically similar in this problem.

As for SGE and its variants, we note that SGE shows better growth at the beginning of the evolutionary process, having the best fitness for over 30 generations. It is then surpassed by Co-PSGE, with presenting statistical differences (Table 5.3). PSGE has a slightly slower growth, ending with results similar to SGE, with no significant differences.

Looking at the results concerning the Pagie polynomial (Fig. 5.3), we observe that all methods perform better than GE, with their curves decreasing faster and ending with a large difference. This is confirmed by the statistical analysis, where all methods present significant differences in comparison to GE (Table 5.2). The method that stands out the most in this problem is PGE, since it is the one that ends with the best performance. On the other hand, we highlight negatively PSGE, because it presents statistical differences in relation to SGE (Table 5.3), and looking at the curve, it means that SGE has better performance in this problem. Co-PSGE does not present statistical differences in relation to SGE, and we can say that in this problem, they are similar, although SGE ends with a better value.

Fig. 5.4 shows the results for the Boston Housing problem. Looking at the training results (Fig. 5.4a), it is possible to see that the fitness of SGE, Co-PGE and Co-PSGE individuals

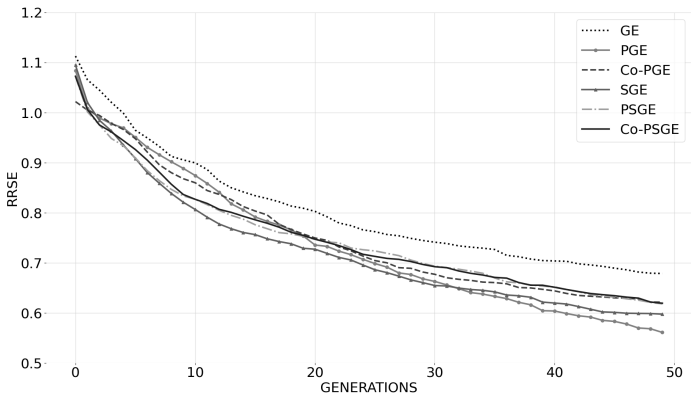
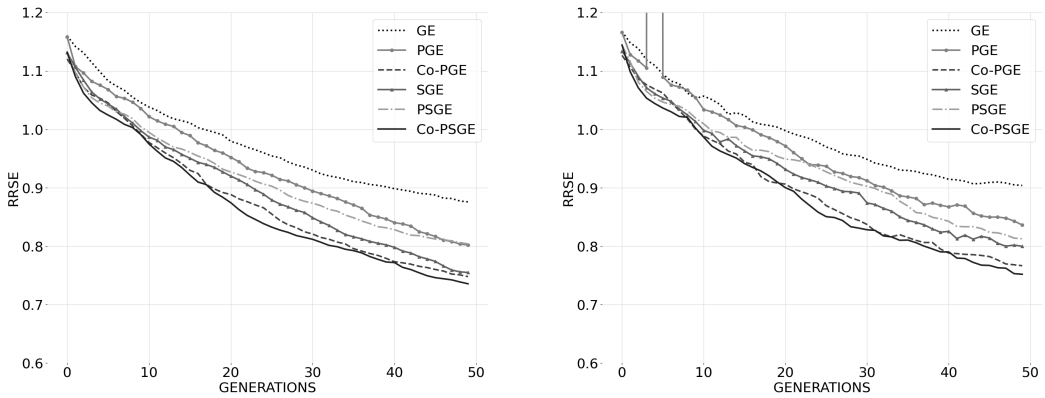


Figure 5.3: Performance results for the Pagie Polynomial. Results are the mean best fitness of 100 runs.



(a) Results for Boston Housing - Training

(b) Results for Boston Housing - Testing

Figure 5.4: Performance results for the Boston Boston Housing problem. Results are the mean best fitness of 100 runs.

rapidly decrease and continue too over the entire evolutionary process, with no statistical differences among them (Table 5.3). The performance of GE is in line with what we observed previously (Fig. 5.2 and 5.3), i.e., a slow decrease on the fitness. As for the PGE and PSGE results, we observe that these are better than those of GE, but worse than the others. This analysis is confirmed by the statistical results, which show that in the training, all methods present statistical differences from GE (Table 5.2).

Even though the training results are important to understand the behaviour of the methods, the testing results are more relevant, because they allow us to evaluate the generalisation ability of the models evolved by each approach. Looking at the test results (Fig. 5.4b) we can see that Co-PGE and Co-PSGE are building models that can generalise better to unseen data. Looking at the statistical tests on Table 5.3, we see that Co-PGE remains statistically similar to SGE, however Co-PSGE shows statistical differences, outperforming SGE. Looking at the graph we notice that SGE is slightly worse in the test than in the training, keeping the statistical differences between PGE and being statistically similar to PSGE (Table 5.3).

Fig. 5.5 shows the evolution of the mean best fitness of the individuals for the 5-bit even

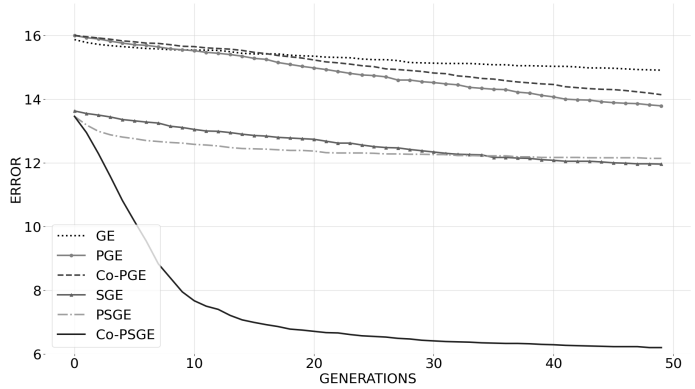


Figure 5.5: Performance results for the 5-Bit Even Parity problem. Results are the mean best fitness of 100 runs.

parity problem. At first, we see that Co-PSGE behaviour stands out, managing to obtain a fitness twice as good as the other methods in fewer generations, presenting statistical differences with GE and SGE (Tables 5.2 and 5.3). The curve has a large slope until generation 10, and then stabilizes. Looking at the results of GE, PGE and Co-PGE, we observe that they start with worse fitness than SGE and its variants. PGE and Co-PGE have a slightly faster decrease, ending with better fitness and being statistically better than GE (Table 5.2). PSGE decreased slightly faster than SGE, however by the 30th generation, it was caught up by SGE, having similar performance, presenting no statistical differences.

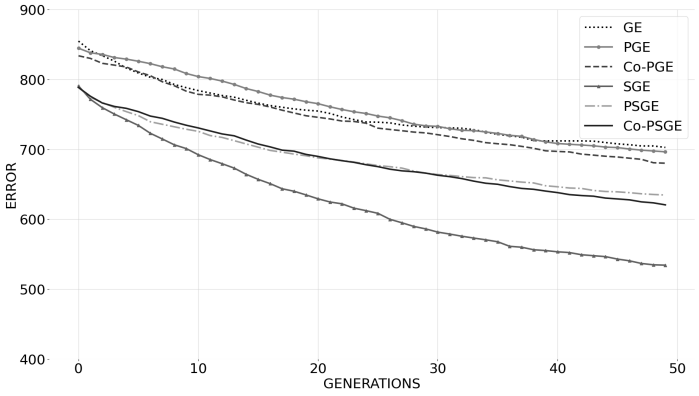


Figure 5.6: Performance results for the 11-bit Boolean Multiplexer problem. Results are the mean best fitness of 100 runs.

The results for the 11-bit Multiplexer are presented in Fig. 5.6. GE, PGE and Co-PGE have a very similar decrease, and initially PGE has a slightly worse behaviour, recovering along the evolutionary process, ending with values very close to the GE, showing no statistical differences (Table 5.2). The Co-PGE has a growth similar to that of GE, gradually moving away to better values with statistical differences. For the remaining methods, the behaviour was different. We see that SGE stands out, having a much more accentuated decrease, with statistical differences in relation to its two variants, Co-PSGE and PSGE. These two methods have a very similar behaviour, which despite not approaching the exceptional performance of SGE, is still statistically better than GE, PGE and Co-PGE (Table 5.3).

It is interesting to note that the No Free Lunch Theorem [48] is reflected in the behaviour observed in some problems by the approaches. This is the case for PGE on Pagie Symbolic Regression, Co-PGE on 5-bit Parity and SGE on 11-bit Multiplexer. The theorem is stated in the original paper by Wolpert et al. [48] as "for any algorithm, any elevated performance over one class of problems is offset by performance over another class", which can be interpreted as that no algorithm is equally good on all problems.

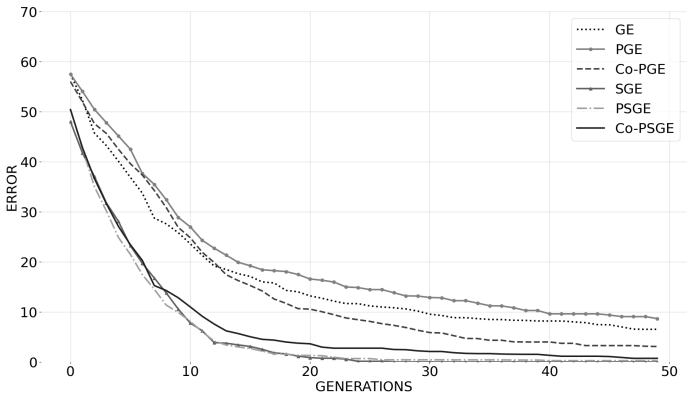


Figure 5.7: Performance results for the Santa Fe Artificial Ant problem. Results are the mean best fitness of 100 runs.

The last analysis studies the performance on the Santa Fe Ant problem. We can observe that SGE, PSGE and Co-PSGE quickly decrease their fitness, diverging from GE, PGE and Co-PGE. Looking at the statistical tests in Table 5.3, we see that PSGE and SGE are statistically similar, and Co-PSGE shows statistical differences from SGE, being slightly worse, with a p-value of 0.044. As the p-value is very close to the significance level ($\alpha = 0.05$), we see that the differences between these methods are minimal. PGE and Co-PGE have no statistical differences with GE (Table 5.2), however, looking at the curve of the graph, we can see that Co-PGE ends up with better average fitness than PGE and GE.

5.2 The Usefulness of Grammars

In this section, we will analyse the evolution of the probabilities of the Probabilistic Context-Free Grammar (PCFG) for each problem. The choice of grammars in grammar-based GP methods plays an important role in the algorithm's behaviour [5, 11], since it defines the search space for the solutions, so this study gives a first insight into the impact of grammars' update on the proposed algorithms.

The methods proposed in this work, present different ways to modify the probabilities of a PCFG. By evolving the population together with the grammar, we are introducing bias with respect to some production rules, guiding the exploration of the search space. We have already seen in the previous section that updating the productions across generations improved the quality of solutions and speed of convergence, in some problems.

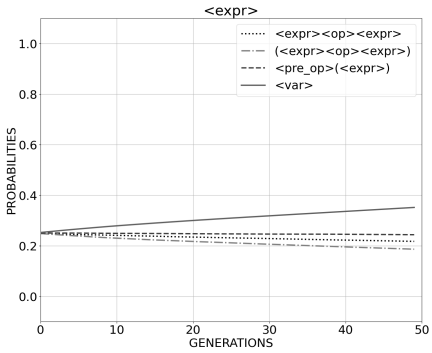
PGE and PSGE evolve the probabilities of the PCFG alternating between using the production rules of the best individual of the current generation and the best overall. Each production of the grammar is updated taking into account the number of times it has been expanded by the individual, the number of times its non-terminal symbol has been expanded and a learning factor, λ. The learning factor is used to make the transitions in

the search space smoother.

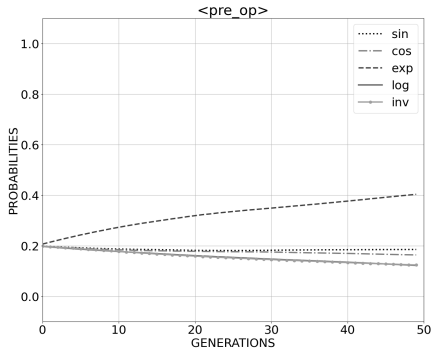
In Co-PGE and Co-PSGE, each individual co-evolves with a grammar. At each generation the individuals are prone to the processes of variation, and also the grammar can mutate.

5.2.1 Probabilities Evolution

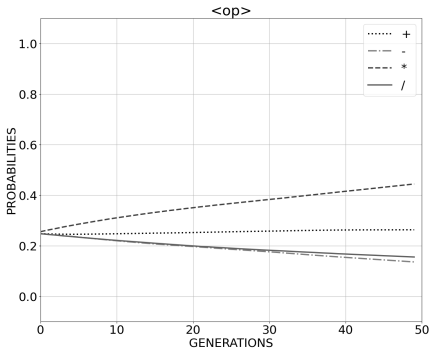
In this section we present an analysis on how the probabilities of certain derivation rules progress over the generations for each problem. This analysis will give us insights into what are the rules that the methods consider more relevant to create better solutions. The values presented are the average of the probabilities of the PCFG at each generation of the 100 runs. The grammars were evolved using PGE, with a learning factor of 1.0%.



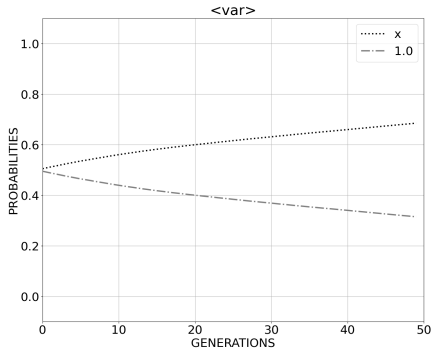
(a) Evolution of grammar probabilities of non-terminal $< expr >$.



(b) Evolution of grammar probabilities of non-terminal $< pre\_op >$.



(c) Evolution of grammar probabilities of non-terminal $< op >$.



(d) Evolution of grammar probabilities of non-terminal $< var >$.

Figure 5.8: Evolution of grammar probabilities of non-terminals used for the Quartic Polynomial. Results are averages of 100 runs.

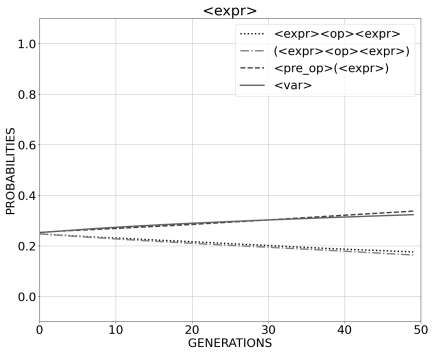
The evolution of probabilities for the quartic problem is shown in Fig. 5.8. In the non-terminal $< expr >$, all production rules end up with different probabilities, with the largest growth in the $< var >$ production rule ($\approx 24\%$). The grammar used (Grammar 5.1) has three recursive and one non-recursive production rules for the non-terminal $< expr >$ so is considered explosive [5], as it has more recursive productions than non-recursive ones. Nicolau et al. [5] showed that explosive grammars have a tendency to create very small individuals by choosing terminals too early, or choose many non-terminal symbols in a row, which can lead to many large and/or invalid individuals. By inserting bias in the non-recursive rule ($< var >$) (Fig. 5.8a), the algorithm is evolving towards balancing the

grammar. Harper [11] showed that balanced grammars produce more diverse individuals, and in [5] Nicolau et al. showed that the use of balanced grammars results in performance improvements.

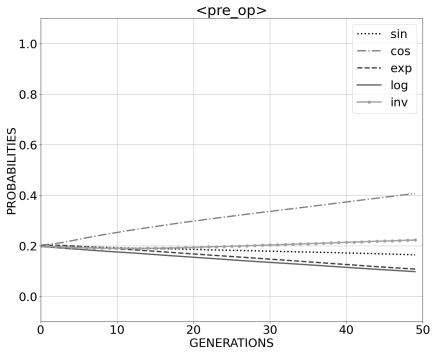
Looking at the evolution of the probabilities for the remaining non-terminal symbols, we see that there are productions more prominent than others. In the case of the non-terminal $< pre\_op >$, we see that the terminal *exp* has a greater change in its probability, ending with approximately 40.3%. The remaining productions end with 18.6% in the case of *sin*, 16.4% in the case of *cos*, 12.3% for *log*, and lastly, 12.4% for *inv*. The fact that the terminal *exp* has such a difference from the remaining productions can imply that, although this token does not belong to the optimal solution, it is favourable for the creation of better solutions.

In what concerns the non-terminal $< op >$, we see that there are two operators that stand out, namely multiplication ($*$) and sum ($+$), which end with 44.5% and 26.4% respectively. The trend for these symbols is in line with what was expected, since it is possible to reach the optimal solution using only these two. The division symbol ($/$), and the subtraction ($-$) end with 15.5% and 13.6%, respectively.

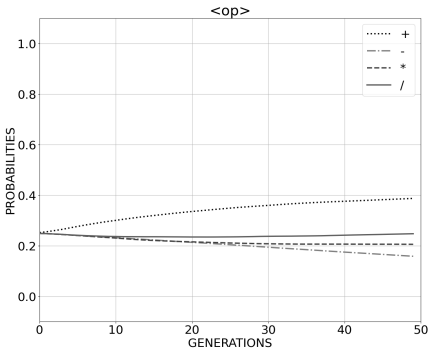
The remaining non-terminal is $< var >$, which has only two productions, has one symbol that stands out, x ($\approx 68.5\%$), as expected, since the other, 1.0 ($\approx 31.5\%$) is not needed to obtain the optimal solution.



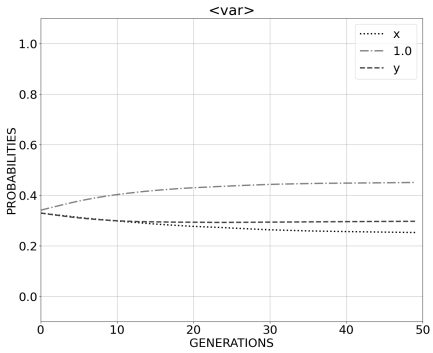
(a) Evolution of grammar probabilities of non-terminal $< expr >$.



(b) Evolution of grammar probabilities of non-terminal $< pre - op >$.



(c) Evolution of grammar probabilities of non-terminal $< op >$.



(d) Evolution of grammar probabilities of non-terminal $< var >$.

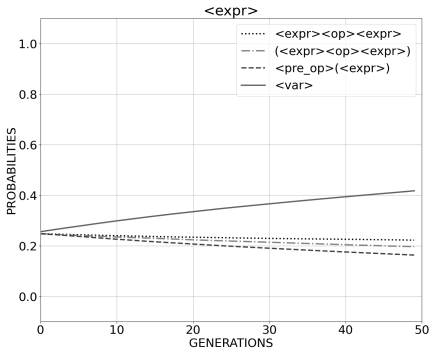
Figure 5.9: Evolution of grammar probabilities of non-terminals used for the Pagie Polynomial. Results are averages of 100 runs.

For the Pagie Polynomial, Fig. 5.9 presents the evolution of the PCFG's probabilities for the different non-terminals over the generations. Looking at the graphs, we can see that different production rules have different growths. Starting with the analysis of the rules of the symbol $< expr >$ (Fig. 5.9a), we notice that production rules that contain two recursive options have much lower probability ($\approx 17\%$) than the remaining two productions ($\approx 33\%$).

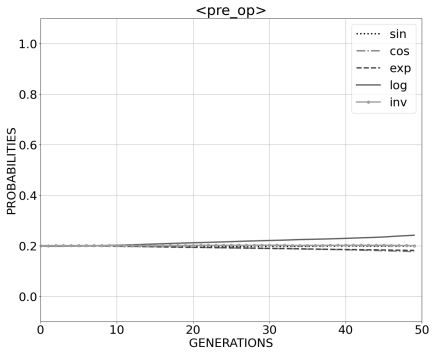
Fig. 5.9c, presents the results for the non-terminal $< op >$, and we can see that the probabilities associated with the operators needed to solve the problem are higher, with the sum ($+$) standing out with about 39%, followed by division ($/$) with 25%, multiplication ($*$) with 21%, and lastly, subtraction ($-$) with 15%.

As observed in the quartic problem, although not required for the optimal solution, there is a production of the non-terminal token $< pre\_op >$ that stands out. In this case, it is *cos*, which ends approximately with 40.7%, with the remaining symbols, *inv*, *sin*, *exp* and *log*, ending with 22.3%, 16.4%, 10.8%, 9.8%, respectively.

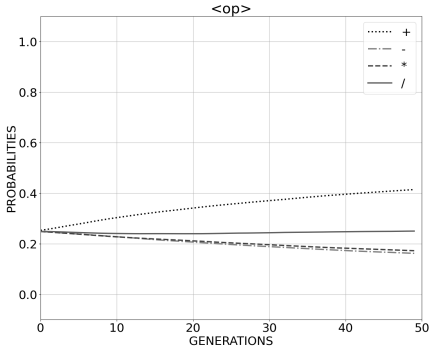
In the non-terminal $< var >$, at the end of the evolution, the different symbols end with different probabilities, however we notice that their evolution is more accentuated in the first 10 generations, and then they stabilize, almost not undergoing changes. In this case, the symbol with the highest probability is 1.0 ($\approx 45\%$), followed by y ($\approx 29.7\%$) and x ($\approx 25.3\%$).



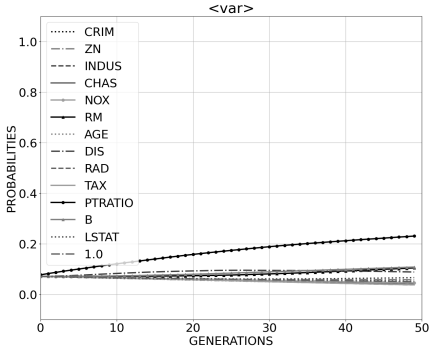
(a) Evolution of grammar probabilities of non-terminal $< expr >$.



(b) Evolution of grammar probabilities of non-terminal $< pre - op >$.



(c) Evolution of grammar probabilities of non-terminal $< op >$.



(d) Evolution of grammar probabilities of non-terminal $< var >$.

Figure 5.10: Evolution of grammar probabilities of non-terminals used for the Boston Housing. Results are averages of 100 runs.

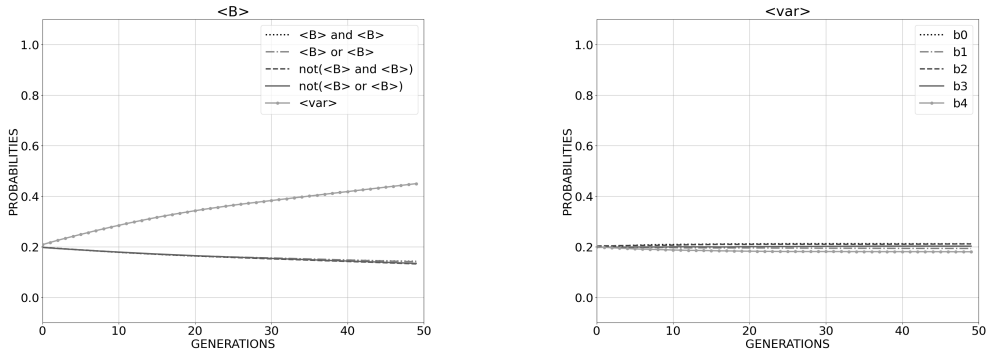
Concerning the Boston Housing, the progression of the probabilities are depicted in Fig. 5.10. The probabilities of the rules of the non-terminal symbol $< expr >$ show very similar behaviour to that analysed by the Quartic Symbolic Regression (Fig. 5.8a). The non-recursive production has a sharp increase ($\approx 42\%$), while the remaining production rules decrease slowly.

Contrary to the other problems of symbolic regression, for the Boston Housing there is not much difference between the different symbols of the non-terminal $< pre\_op >$, and at the end there is a slight increase of the *log* probability, which ends with 24.2%. The remaining outputs have very close values, with *sin* and *inv*, having approximately 20% each (which is the initial value), followed by *exp* with 18.1% and finally *cos* with 17.7%.

In relation to the symbol $< op >$ we see that the terminal of sum (+) and division (/) stand out, with 41.5% and 25%, respectively, and multiplication ($*$) and subtraction ($-$), end with 17.3% and 16.2%, respectively.

The results of the probabilities for the derivation options of the non-terminal $< var >$ are the most interesting because it contains the features that describe the problem. Looking at the evolution of the probabilities associated with each production, we can understand which of these features are more relevant to accurately predict the price of houses. Looking at the results (Fig. 5.10d), one can see that $PTRATIO$ stands out in terms of the probability of being selected. $PTRATIO$ represents the pupil-teacher ratio by town. This is in line with the results reported by Che et. al. [49]. Another interesting result is to see that the feature RM (the third most important feature in [49]), which is the average number of rooms per dwelling, is also on the top three of our results.

These results confirm not only the relevance of these features to the Boston Housing problem, but also allow us to perform feature analysis and provide an explanation to the results achieved. This means that at the end of the evolutionary process one can look at the final distribution of the probabilities in the grammar, and analyse the relative importance of each production and derivation rules, and see how they are used to create the best models.

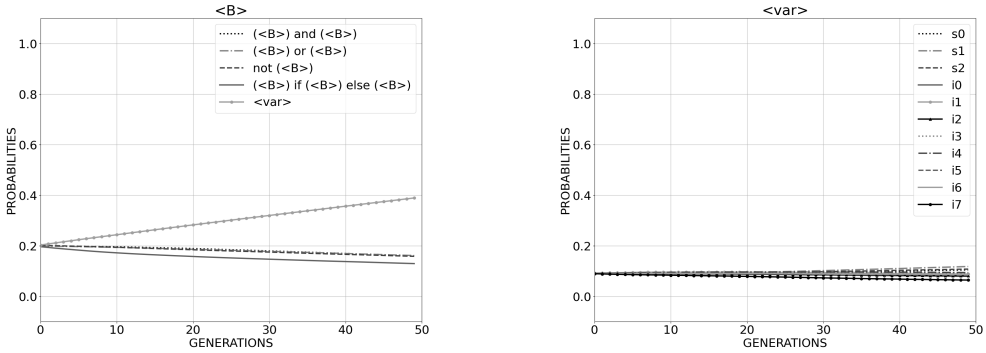


(a) Evolution of grammar probabilities of non-terminal $< B >$.

(b) Evolution of grammar probabilities of non-terminal $< var >$.

Figure 5.11: Evolution of grammar probabilities of non-terminals used for the 5-bit Parity. Results are averages of 100 runs.

Looking at the evolution of the probabilities of the non-terminal $< B >$ of both 5-bit Parity Problem (Fig. 5.11a) and the 11-bit Multiplexer (Fig. 5.12a), we can again observe a biased growth of the non-recursive production, $<var>$ ($\approx 45\%$, for 5-bit Parity, and $\approx 39\%$, for 11-bit Multiplexer). In 5-bit Parity, the remaining productions decrease to approximately 13.5%, while in the 11-bit Multiplexer there is a small difference in the



(a) Evolution of grammar probabilities of non-terminal $< B >$.

(b) Evolution of grammar probabilities of non-terminal $< var >$.

Figure 5.12: Evolution of grammar probabilities of non-terminals used for the 11-bit Multiplexer. Results are averages of 100 runs.

probabilities of the remaining productions. Three of the productions, which contain one or two recursive rules, decrease to approximately 16%, while the other rule, which contains three recursive rules, decreases to 13%. Once again we can observe an adjustment of the probabilities in order to balance the grammar.

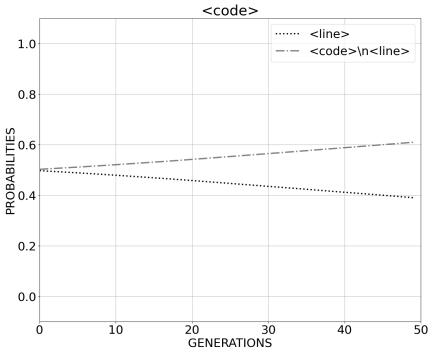
As for the non-terminal $< var >$ of 5-bit Parity (Fig. 5.11b), there is little variation between the different probabilities, at the end of the 50th generation $b4$ has the lowest value, with about 18%, and the highest belonging to $b0$ and $b2$, with about 21% each. Because there are no meaningful differences in the probabilities of these terminal symbols, it could mean that they all have the same impact on the quality of the solutions generated.

In the case of the 11-bit multiplexer (Fig. 5.12b), there is also little discrepancy between the different variables, however, the $s[0..2]$ variables are slightly higher, the highest being $s1$ with approximately 11.8% and the lowest being $s0$ with around 10.4%. Within the $i[0..7]$ variables, the highest is the $i4$ with 9.4% and the lowest is $i7$ with 6.4%.

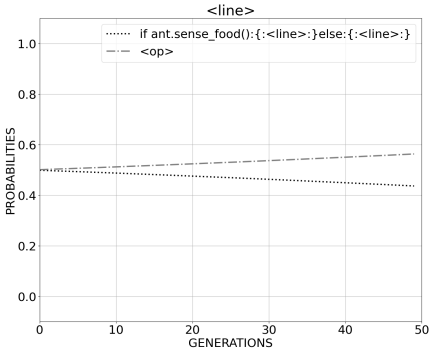
In both problems there is little difference between the probabilities of the different variables, which may mean that none excels in creating better solutions, or else it may need more generations and/or a different learning factor, to be able to introduce bias in the rules. It is also important to note that in the performance analysis, in the 11-bit problem, there were no significant differences between GE and PGE, showing that the evolution of probabilities for this problem does not result in a improvement of performance, with the parameters used.

Before proceeding to the analysis of the probabilities of the different non-terminals of the Santa Fe Ant problem, it is important to highlight that in this problem, PGE had no significant differences with respect to GE, that is, the updating of the probabilities did not make the exploration of the search space more effective. It is also interesting to note, that the grammar used in this problem, unlike the grammars used in the other problems, is not considered explosive, because the non-terminal symbols have the same number of recursive and non-recursive productions. Additionally, this grammar contains two non-terminal symbols with recursive options, $< code >$ and $< line >$.

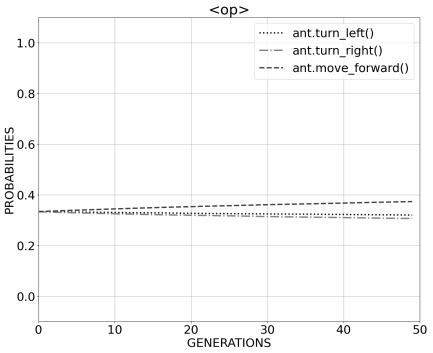
For the non-terminal $< code >$ (Fig. 5.13a) we observe a different behaviour of the evolution of probabilities, compared to the other problems, as the production containing a recursive rule ($< code > \backslash n < line >$) ends with higher probability ($\approx 61\%$) than the



(a) Evolution of grammar probabilities of non-terminal $< code >$.



(b) Evolution of grammar probabilities of non-terminal $< line >$.



(c) Evolution of grammar probabilities of non-terminal $< op >$.

Figure 5.13: Evolution of grammar probabilities of non-terminals used for the Santa Fe Ant Trail. Results are averages of 100 runs.

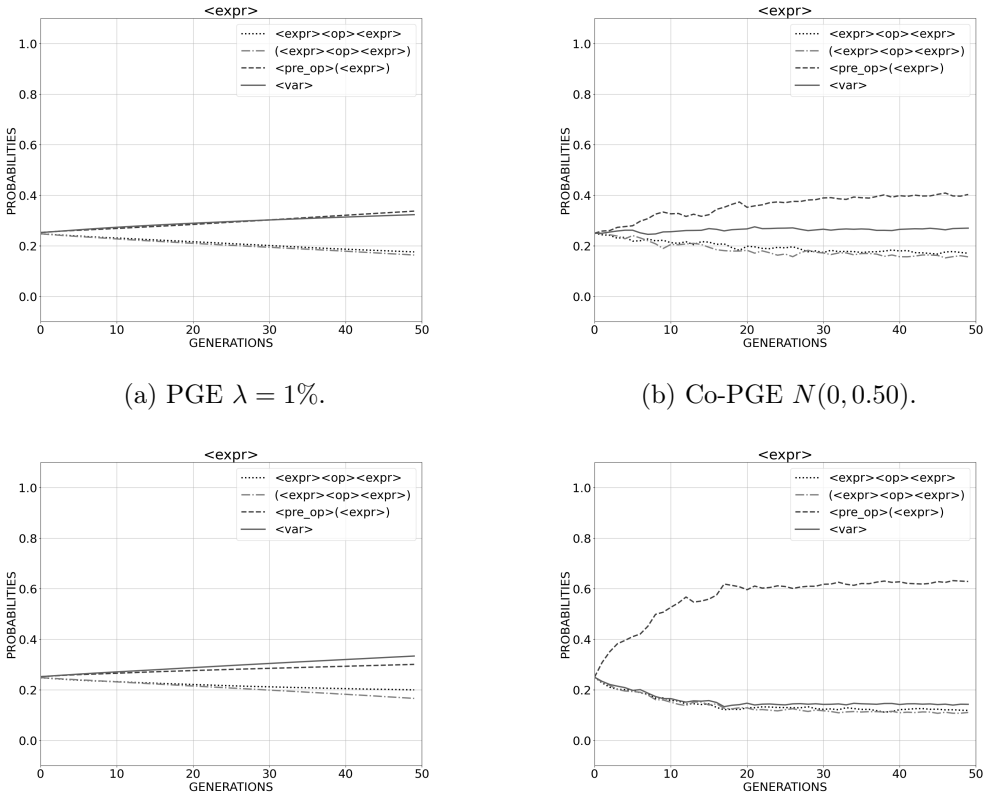
non-recursive production ($< line >$, with $\approx 39\%$). Langdon et al. [50] showed, through the generation of random solutions, that the average fitness of the solutions increases with their size, while the variance decreases. In the paper by Wilson et al. [51], was studied the relationship between the program generated to solve the trail, its behaviour and the fitness of the solutions evolved for the Santa Fe Ant problem. The analysis showed that the least prevalent approach followed by the solutions is the one that follows the optimal path, and one of the most prevalent approaches followed by the solutions, is to visit the largest number of cells, which eventually leads to find more food. This last approach may be the reason why PGE favours the recursive production of the non-terminal $< code >$. By choosing recursive productions more often, it makes it more likely that larger individuals will be generated, thus visiting more cells.

In the case of the non-terminal $< line >$ (Fig. 5.13b) we find that the non-recursive production has a higher growth, ending with 56% while the non-recursive one ends with 43%. This may be a mechanism to prevent bloat from occurring in the solutions. Finally, looking at the graph concerning the non-terminal $< op >$ (Fig. 5.13c), we see that there is a slow growth, with the production of moving forward ending with higher probability ($\approx$ 37%), while the productions concerning the left turn and the right turn end with approximately 32% and 31%, respectively.

To complete this analysis of the evolution of grammars, we show the evolution of probabilities using the different proposed methods. As showing the evolution for all non-terminals

with all methods would be too exhaustive and extensive, we selected some non-terminals in which we observe more differences in the evolution of probabilities.

The graphs for PGE and PSGE show the average PCFG probabilities of the 100 runs, evolved with a learning factor, λ of 1%. In the graphs of the evolution of the Co-PGE and Co-PSGE probabilities, the values shown are the average of the grammars of the best individual of each generation, from the 100 runs, and were evolved using a probability of occurring a mutation of 5% and the Gaussian mutation of $N(0, 0.50)$.



(a) PGE $\lambda = 1\%$.

(b) Co-PGE $N(0, 0.50)$.

(c) PSGE $\lambda = 1\%$.

(d) Co-PSGE $N(0, 0.50)$.

Figure 5.14: Evolution of the non-terminal $< expr >$ of the Pagie Polynomial, using different methods. Results are averages of 100 runs.

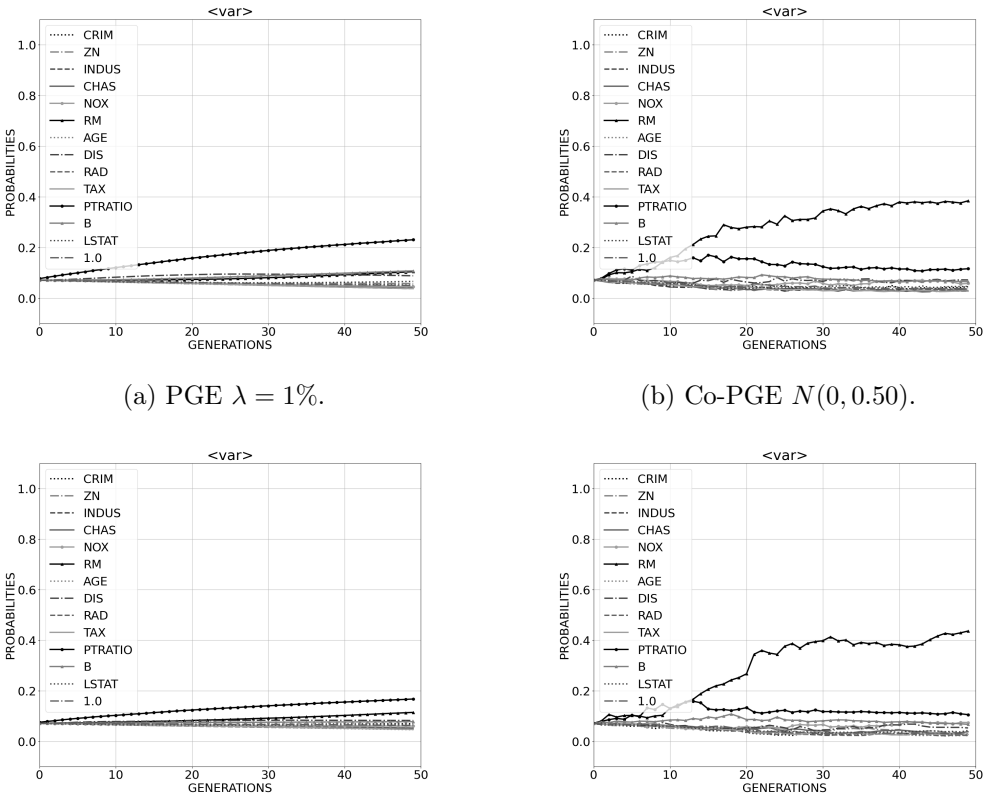
The first non-terminal selected was the $< expr >$ of the Pagie polynomial (Fig. 5.14). Looking at the graphs of the evolution of probabilities, we see that there are two rules, $< expr >< op >< expr >$ and $(< expr >< op >< expr >)$, which are the ones that contain two recursive symbols, that end with less probability (with approximately the same value) in all methods. As we saw in the previous analyses, this may be a mechanism to prevent bloat from occurring in individuals. The biggest difference in the adjustments of probabilities between the methods is in the remaining productions: $< var >$ and $< pre\_op > (< expr >)$.

In this problem we saw that all the proposed methods present statistical differences in relation to GE (Table 5.2). Despite not surpassing statistically the performance of SGE, PGE finished with the best mean best fitness (Fig. 5.3). In Fig. 5.14a and 5.14c, referring to PGE and PSGE, respectively, we see that the evolution of probabilities between the two methods is very similar. In PSGE the non-terminal $< var >$ ends with slightly higher probability than $< pre\_op > (< expr >)$, with 33% and 30%, respectively, while in PGE it is the non-terminal $< pre\_op > (< expr >)$ that ends with higher probability, with 34%

while $< var >$ ends with 32%.

The co-evolutionary approaches present a very different evolution. In the case of Co-PGE (Fig. 5.14b), the rule with only one recursive option, $< pre\_op >$ ($< expr >$) stands out right from the beginning, ending with about 40%, followed by $< var >$ with 27%, and the remaining productions with 17% and 16%. In the case of Co-PSGE (Fig. 5.14d), we see a big increase in the probability of selecting $< pre\_op >$ ($< expr >$), in the first 20 generations, then stabilizes, ending with 63%. The production $< var >$ ends with 14%, slightly more than the remaining productions $< expr >< op >< expr >$ and ($< expr >< op >< expr >$), which end with 12% and 11%, respectively.

In spite of following different approaches to evolve the probabilities of the grammar, none of the methods was able to surpass the performance of SGE in the Pagie Symbolic Regression problem.



(a) PGE $\lambda = 1\%$.

(b) Co-PGE $N(0, 0.50)$.

(c) PSGE $\lambda = 1\%$.

(d) Co-PSGE $N(0, 0.50)$.

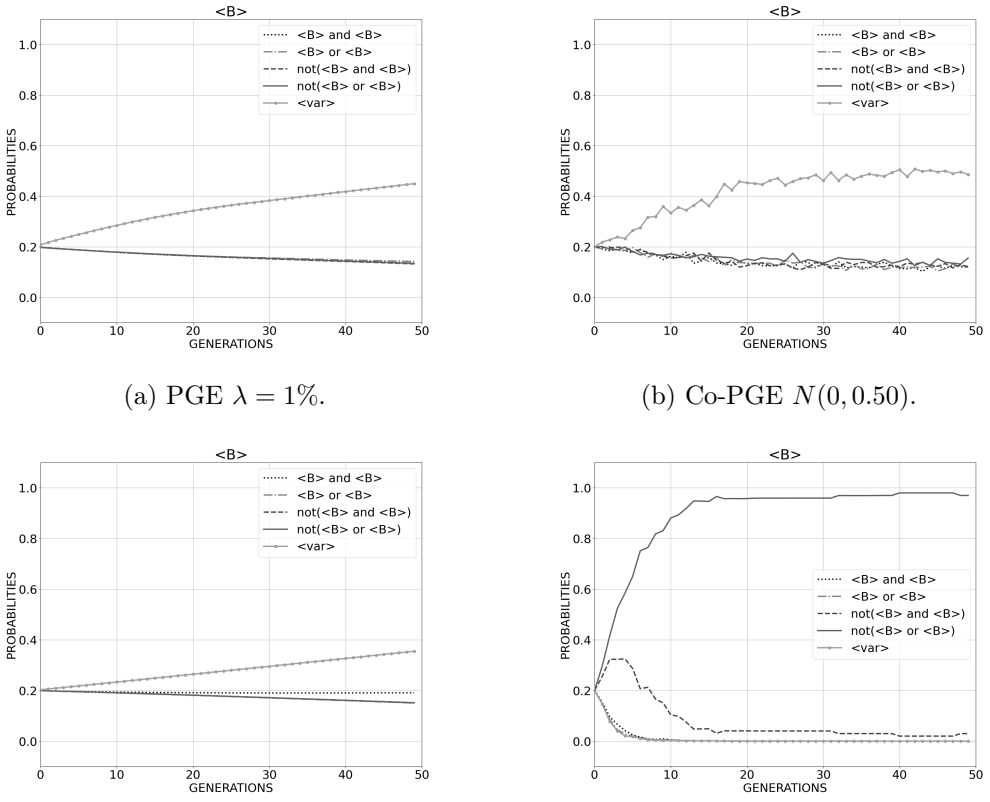
Figure 5.15: Evolution of the non-terminal $< var >$ of the Boston Housing problem, using different methods. Results are averages of 100 runs.

For the Boston Housing problem we selected the non-terminal $< var >$ because it is interesting to see which features were considered more relevant to generate better solutions, among the different methods. The evolution of the probabilities using the different proposed methods is depicted in Fig. 5.15. In this problem all methods were statistically better than GE (Table 5.2) in test and training, with Co-PSGE being statistically better than SGE in the test data (Table 5.3).

We can see that the approach followed by PGE (Fig. 5.15a) and PSGE (Fig. 5.15c) is quite similar, in which the $PTRATIO$ feature ends with higher probability, followed by RM. In PGE there is a slightly larger difference between the final probabilities than in

PSGE.

In the case of the co-evolutionary strategies, there is a much higher prominence of the *RM* feature, ending with 38% in Co-PGE (Fig. 5.15b) and with 44% in Co-PSGE (Fig. 5.15d). This prominence in the *RM* variable by these two methods might be advantageous to generate better individuals, since in the training results, the performance of the two methods was statistically similar compared to SGE, and in the test results, Co-PSGE managed to overcome the performance of SGE, with statistical differences.



(a) PGE $\lambda = 1\%$.

(b) Co-PGE $N(0, 0.50)$.

(c) PSGE $\lambda = 1\%$.

(d) Co-PSGE $N(0, 0.50)$.

Figure 5.16: Evolution of the non-terminal $< B >$ of the 5-bit Parity problem, using different methods. Results are averages of 100 runs.

Lastly, we selected the non-terminal $< B >$ of the 5-bit Parity problem. We can observe through the graphs of Fig. 5.16, that the evolution of the probabilities of the productions is quite similar between GE, PGE and Co-PGE, being that the non-recursive production, $< var >$, ends with higher probability, and the remaining productions with similar probabilities among them.

In this problem the behaviour of Co-PSGE stood out (Fig. 5.5), being statistically better than SGE, and we can see through the evolution of the probabilities on Fig. 5.16d that it presents a different behaviour in comparison to the other approaches: the probability of the rule $not(< B > or < B >)$ increases very quickly, stabilizing around generation 15 and ends with about 96%, while the rule $not(< B > and < B >)$ ends with 3%, and the others with a value very close to 0%. In the performance analysis we also saw the mean best fitness of the solutions decreasing rapidly in the first 15 generations and then decreasing more slowly. The most prominent rule has two recursive productions which encourages the creation of very large individuals.

In both the Pagie and 5-bit Parity analysis, we saw that Co-PSGE introduced too much bias in a recursive production, which leads to the creation of large individuates. However, while in PGE and Co-PGE bias in recursive productions can lead to the creation of many invalid individuals (because the solutions are limited by the genotype size, and the codons may end before creating a valid individual), in PSGE and Co-PSGE this does not happen because the genotype is dynamic and, in case the depth limit is reached, only non-recursive productions can be chosen, creating a valid individual.

In the 5-bit Parity problem, Co-PSGE (Fig. 5.16d) evolved the grammar so that the only non-recursive option of the non-terminal $< B >$ ($< var >$) ends with probability of being selected very close to zero. When the depth limit is reached this is the only rule that can be chosen, so even though it has probability close to zero it does not prevent the rule from being chosen.

5.3 Representation Analysis

To study the redundancy and locality of the different methods, we resort to the metric Mutation Innovation (MI) which represents the phenotypic distance between two individuals. This distance will be calculated using the dynamic programming algorithm proposed by Zhang et al. [52] in which three types of operations are considered: edit (in which the label of a node is modified), remove (a node is eliminated and its children become children of the parent) and insert (a node is added to the tree, and the children of its parent node become its children). This study is based on the work developed by Raidl et al. [26] and Lourenço et al. [16].

In this analysis we will consider GE, SGE, PGE and PSGE. The proposed approaches will be studied with and without updating the probabilities, to allow an analysis of the proposed representation without the biased grammar interfering. Note that without updating the probabilities, the algorithm becomes the same as the co-evolutionary versions if no mutations are performed on the grammars used to generate each individual, because both approaches use the same individual's representation. Moreover, we do not consider co-evolutionary variants due to the stochasticity that exists simultaneously in the mutations in the individuals and in the grammar used to map each individual.

The parameters used for this analysis are the same as those presented in Table 5.1. In the graphics where the probabilities of the PGE and PSGE grammars are updated, a learning factor of 5% was used. Regarding the mutation, for GE, the selected codon is replaced by a new generated one in the interval $[0, 255]$, for PGE is performed a float mutation, in which the codon is replaced by a new one generated in the interval $[0, 1]$. In the case of SGE, the mutated codon is replaced with a different valid option, while in PSGE, a Gaussian mutation with $N(0, 0.50)$ in the codon value is performed.

This analysis was done for the Quartic polynomial of Symbolic Regression using Grammar 5.6.

5.3.1 Redundancy

Redundancy is related to how changes in genotype result in the same phenotype. The GE is known for having high redundancy [4], which means that in most cases genotypic changes result in the same phenotype.

$\langle start \rangle$ ::= $\langle expr \rangle$

$\langle expr \rangle$ ::= $\langle expr \rangle$ $\langle op \rangle$ $\langle expr \rangle$ | ($\langle expr \rangle$)
| $\langle pre\_op \rangle$ ($\langle expr \rangle$) | $\langle var \rangle$

$\langle op \rangle$ ::= + | - | * | /

$\langle pre\_op \rangle$ ::= sin | cos | exp | log

$\langle var \rangle$ ::= x | 1.0

Grammar 5.6: Symbolic Regression grammar for locality and redundancy analysis.

Redundancy will be studied by analysing the frequency of mutations that do not generate a phenotypic change, that is, whose MI is equal to zero. For this, 10000 repetitions were made, in which 20 successive mutations were applied to a randomly generated individual. After each mutation, we compute the phenotypic distance between the new individual and the original. To better analyse the results, we created an heatmap, where the values of the distances were placed in 14 intervals on the vertical axis [16], and the 20 mutations, k, on the horizontal axis. For each position in the graph there is a shaded square, which represents the percentage of mutated individuals whose MI is in the respective interval range. The higher the percentage, the darker the square.

Regarding the analysis made to PGE and PSGE, a learning factor of 5% was used with the best individual resulting from each step, k, used to updated the PCFG probabilities. After suffering the mutation and the grammar updated, the individuals were remapped.

Fig. 5.17 and 5.18 show the average of the 10000 repetitions, obtained over 20 consecutive mutations, for GE and PGE. From the graphs we can observe that the MI values in both methods are very similar. In both methods the percentage of mutations that did not result in phenotypic changes after a mutation has occurred in the genotype is 92%. This value is in line with the analysis made by Lourenço et al. [24]. This percentage decreases as more mutations occur, but at the end of 20 mutations this value is still high, being 50.1% in the case of GE and 48.9% in the case of PGE.

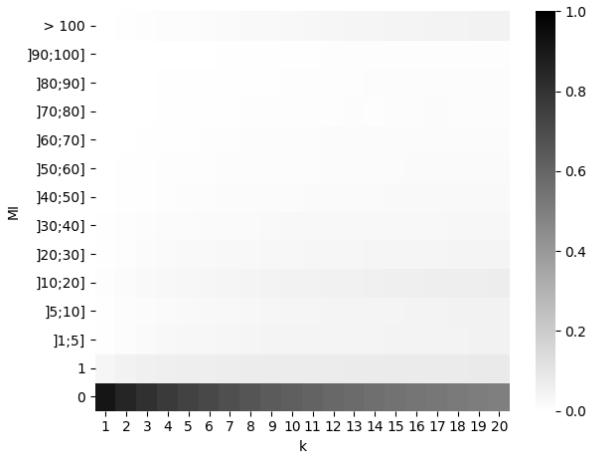
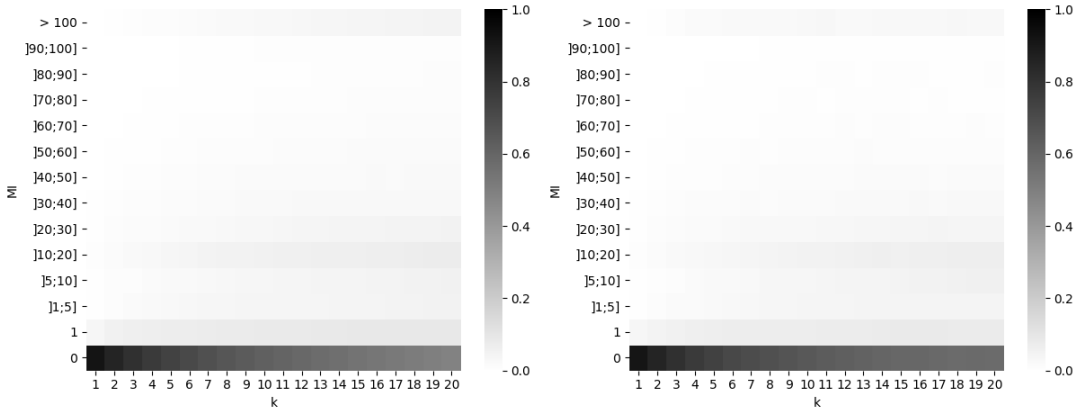


Figure 5.17: Distribution of the phenotypic distances between an original solution and mutants iteratively generated over a random walk with 20 steps, using GE. Results are averages of 10000 runs.



(a) PGE without PCFG update.

(b) PGE with PCFG update ($\lambda = 5\%$).

Figure 5.18: Distribution of the phenotypic distances between an original solution and mutants iteratively generated over a random walk with 20 steps, using PGE. Results are averages of 10000 runs.

The gradient of the colour of the graphs is very similar between the GE graph (Fig. 5.17) and the PGE graph without updating probabilities (Fig. 5.18a), however there is a small difference in the distribution of the gradients with respect to the graph in which the probabilities were adjusted (Fig. 5.18b). At the end of 20 consecutive mutations ($k = 20$), we see that GE and PGE without probability updating have about 5.4% and 5.2%, respectively, of individuals with $MI > 100$, while PGE with update of the grammar has only 2.8%. On the other hand, in GE and PGE without probability updating, about 50.1% and 49.5%, respectively, are non-effective mutations ($MI = 0$), while in the PGE with grammar updating has about 58%. These values are still very high, as it means that half of the solutions remained the same as the initial ones.

Through the performance results, we saw that the update of the PCFG probabilities proved to be valuable for the evolution of solutions. In this analysis, we see that PGE presents slightly more redundancy after executing the 20 mutations. Although this is normally seen as a problem, in this case it might be a feature, since the number of mutations where $MI > 100$ decreased. Having a $MI > 100$ may imply that the trees of the individual are very large, and bigger differences between the k-mutated individual and the original means we lost the exploitation, trading it with exploration.

Looking at the plot of the distribution of phenotypic distances of SGE (Fig. 5.19), we see that it does not show redundancy after a mutation, which is justified since in this algorithm the type of mutation used forces a different production rule to be chosen. In the case of PSGE (Fig. 5.20), we see that approximately 39% of mutations are not effective ($MI = 0$).

Although PSGE does not eliminate redundancy completely, it shows lower redundancy when compared with GE and PGE, showing that the representation used (a set of dynamic lists, with each list containing the probabilities used to choose the productions of each non-terminal) reduces the number of non-effective mutations. Furthermore, as k-mutations are made, the percentage of cases in which $MI = 0$ diminishes gradually, being very close to zero after 5 consecutive mutations.

PSGE also presents higher percentage of k-mutated individuals whose phenotypic changes exceed the distance of 100 ($MI > 100$) when compared to the original individual, which indicates that PSGE is favouring the emergence of larger individuals. In the case of SGE, at the end of 20 mutations it ends with approximately 7% of individuals with $MI > 100$,

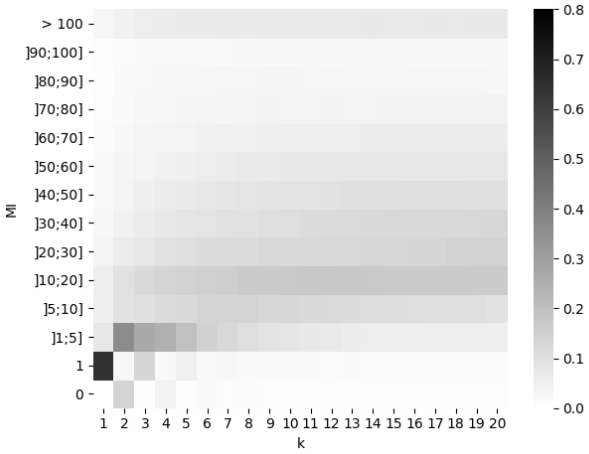
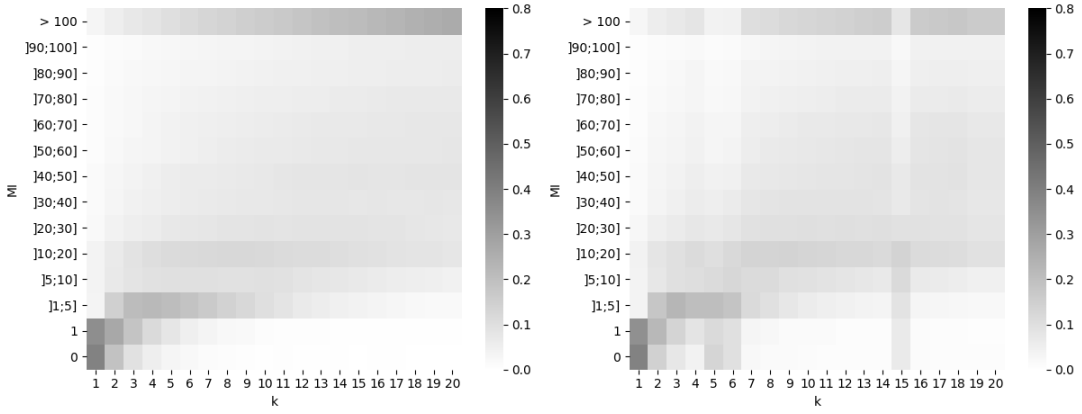


Figure 5.19: Distribution of the phenotypic distances between an original solution and mutants iteratively generated over a random walk with 20 steps, using SGE. Results are averages of 10000 runs.



(a) PSGE without grammar update.

(b) PSGE with grammar update ($\lambda = 5\%$).

Figure 5.20: Distribution of the phenotypic distances between an original solution and mutants iteratively generated over a random walk with 20 steps, using PSGE. Results are averages of 10000 runs.

whereas PSGE without updating probabilities ends with approximately 26.5%. Looking at the PSGE graph in which the grammar was updated (Fig. 5.20b), we see that the percentage of $MI > 100$ is lower when compared to the plot where the grammar was not updated (Fig. 5.20a), as observed in PGE, ending with 16.4%.

It is also important to highlight that in PSGE (Fig. 5.20a) there is a smoother transition in the colour of the gradient in the diagonal, with a higher percentage of individuals with more phenotypic differences in relation to the original individual. By having individuals with different trees sizes, we have variety in the solutions, which results in a better exploration of the search space. In SGE we can see a higher percentage of individuals with phenotypic distances between 10 and 40 ($10 < MI < 40$).

In the PSGE graph with grammar update (Fig. 5.20b), we see a break in the gradient at steps k= 5, 6, and 15. To better understand this behaviour, we repeated the analysis using different seeds, and verified that the breaks are common, and there is no pattern to its occurrence, i.e., it can happen at any step. Although a more detailed analysis is needed to

understand this behaviour, for example, analysing the size of the trees, see the distribution of solutions by the search space at the beginning and compare it with the evolution of the probabilities, we can assume that this behaviour is related to the update of the grammars themselves. Besides the fact that at each step a mutation occurs, the phenotype of the individual may change due to the change of the probabilities of the grammar, for example, if the mutation performed is not effective, the phenotype may change anyway.

In the redundancy analysis of GE (Fig. 5.17), PGE (Fig. 5.18a) and PSGE (Fig. 5.20a) without the grammar update, we highlight that the methods have higher percentage of individuals with phenotypic differences compared to the original (greater than 100). We will now explain why we do not see the same behaviour in SGE.

The first factor to take into consideration, is that the type of mutation used by the methods is different. In GE and PGE the codon is replaced by a randomly generated value within the established range. In PSGE, a Gaussian mutation is performed to the codon, with $N(0, 0.50)$. The mutation of these three methods despite changing the codon value of the genotype, does not prevent the codon from being mapped again into the same derivation rule, hence the redundancy. SGE mutation randomly selects a codon (an integer value representing a derivation rule) and randomly replaces it with a different valid option, resulting in no redundancy. Also note that solutions are limited in all methods. In the case of GE and PGE, they are limited by the size of the genotype, and in the case of SGE and PSGE, they are bounded by the depth limit.

Taking this into account, we can also say that GE, PGE, and PSGE mutation is independent, i.e., the probability of choosing a derivation rule does not depend on the previous derivation rule, and SGE mutation is dependent. The difference in the type of mutation can affect the size of individual's trees and therefore, when mutations occur, cause more differences between phenotypes.

We will now study the probabilities of selecting different production rules, using $p(A)$, to represent the probability of an event A occurring, and $p(A|B)$ to define a conditional probability that refers to the probability of an event A happening, knowing that B occurred. It is also worth noting that $p(A|B) = p(A)$ is true if A and B are independent events. Using the productions of the non-terminal $< expr >$ of the grammar presented as example (Grammar 5.6), let us consider R the event of choosing a recursive production, and $\bar{R}$ the event of choosing a non-recursive production. When generating the individuals for the methods, $p(R) = 75\%$ and $p(\bar{R}) = 25\%$, because the non-terminal $< expr >$ has three recursive and one non-recursive production rules. In case a mutation occurs in the list referring to the non-terminal $< expr >$, the probabilities between the methods will be different. In SGE, we have to consider conditional probabilities, as the new production to be chosen randomly depends on the previous one. Since the previous production cannot be chosen again, we have: $p(R|R) = 2/3 \approx 66\%$, $p(R|\bar{R}) = 100\%$, $p(\bar{R}|R) = 1/3 \approx 33\%$, and $p(\bar{R}|\bar{R}) = 0\%$. In GE, PGE, and PSGE, R and $\bar{R}$ are independent events, because the occurrence of one event, does not affect the probability of the next event, so $p(R|R) = p(R|\bar{R}) = p(R) = 3/4 = 75\%$, and $p(\bar{R}|R) = p(\bar{R}|\bar{R}) = p(\bar{R}) = 1/4 = 25\%$.

In the generation of individuals, the probability of whether or not a recursive production occurs is the same in all methods (because it does not depends on the previous production rule chosen), so the individuals are more likely to have a lot of recursive productions in the phenotype, as these have 75% probability of being expanded. This is verified by examining the graphs (Fig. 5.17, 5.18, 5.19 and 5.19), as at the first step ($k = 1$) the percentage of $MI > 100$ is very similar between the methods: 0.32% for GE, 0.39% for PGE, 3.01% for SGE, and 3.13% for PSGE.

Considering the case in which is selected a production rule of the non-terminal $< expr >$ to mutate, there is 75% probability that the rule is recursive. If the rule is replaced with another recursive production ($p(R|R)$), the tree will either maintain its size or increase, whereas if it is replaced with a non-recursive production ($p(\bar{R}|R)$), its size will decrease. In the case of GE, PGE, and PSGE $p(\bar{R}|R) = p(\bar{R}) = 25\%$. In SGE, $p(\bar{R}|R) = 1/3 \approx 33\%$. SGE has higher probability of reducing the individual's tree, which may result in SGE producing smaller individuals and, therefore, fewer differences between individuals. These results require further research to analyse the size of the trees between the different methods, and other possible causes for this difference in the behaviour.

As for the analysis of PGE and PSGE when updating probabilities (Fig. 5.18b and 5.20b), we saw that the percentage of individuals with $MI > 100$ went from 5.2% to 2.8% in the case of PGE, and from 26.5% to 16.4%, in the case of PSGE. We also showed from the prior performance analysis that these methods present better or equivalent to their variants in most problems. The analysis of updating the probabilities of PGE (Section 5.2) showed that for the explosive grammars, the probabilities evolve to favour non-recursive production rules, balancing the weights of the grammar. We also noticed that, despite a decrease in the percentage of cases when $MI > 100$, the percentage of redundancy increased somewhat, rising from 48.9% to 58.9% in the case of PGE and from 0.2 percent to 1% in the case of PSGE. This behaviour is expected in grammars with only one non-recursive option, because by increasing the probability of this rule being selected, in addition to balancing the bias of the non-recursive and recursive symbols of the grammar, we are also increasing the probability of choosing that production again, hence the slightly increase in the redundancy.

5.3.2 Locality

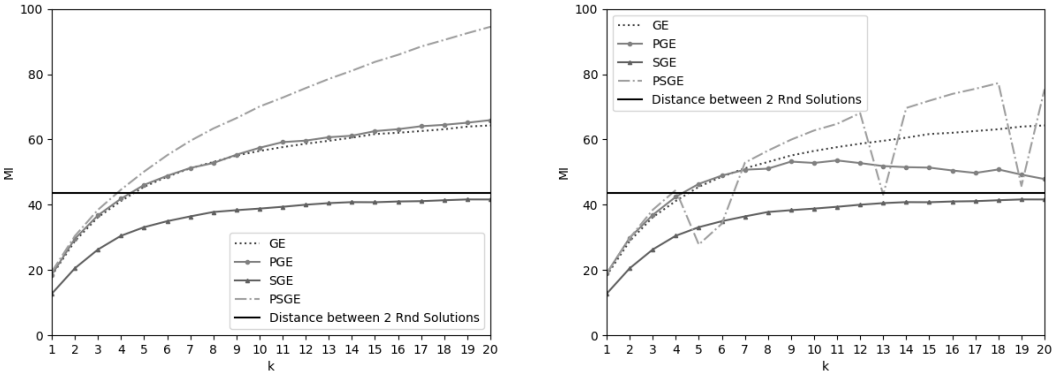
Locality refers to how changes in genotype affect the phenotype. A representation is said to have high locality if small changes in genotype result in small changes in the phenotype, resulting in a better exploration of the search space. It has been extensively studied in the literature that GE suffers from low locality [3].

The analysis of the locality will be done by calculating the MI between a random individual and a individual resulting from a mutation that caused effective changes in the phenotype [26]. This process was repeated 10000 times, and 20 effective mutations ($MI > 0$) were performed iteratively. At each iteration the MI value of the new and original individual was calculated, saving the average of all repetitions.

Two separate graphs were made, one in which PGE and PSGE were run without updating the probabilities of the grammar (Fig. 5.21a), and another in which the probabilities were updated with the best individual of the 10000 at each k-mutation, using a learning factor, λ, of 5% (Fig. 5.21b).

A line representing the average value of 10000 measurements of the phenotypic distance between two randomly generated solutions was placed horizontally to serve as a guide.

By looking at Fig. 5.21a, we notice that the lines for GE and PGE are very similar. After 5 successive mutations ($k = 5$), the solutions start to have higher MI than the average of the phenotypic differences between two random solutions. This behaviour is considered as low locality since the trees already start to be very different. The SGE line presents a smoother MI increase, and at the end of 20 consecutive mutations, this value is below the line that represents the distance between two random solutions. This means that each mutation causes few changes in the phenotype of the individual, resulting in high locality,



(a) PGE and PSGE without PCFG probabilities update.

(b) PGE and PSGE with PCFG probabilities update ($\lambda = 5\%$).

Figure 5.21: Evolution of MI over a random walk with 20 steps, composed just by effective mutations ($MI > 0$). Results are averages of 10000 runs.

which allows more exploitation of the search space.

In the redundancy analysis PSGE had a different behaviour from the other methods, presenting less redundancy than GE and PGE, but higher than SGE, so it was expected that the MI line in the locality analysis would be between the line of SGE and GE, which does not happen. In PSGE the MI has a much more accentuated rise, surpassing the line of the average distance between two random solutions around mutation 4, before GE and PGE. Unlike the other methods, the PSGE line does not seem to stabilize.

The results of the evolution of MI, in which the grammar of PGE and PSGE was updated (Fig. 5.21b), show what we verified in the redundancy analysis: with the updating of the probabilities of the grammar, the MI growth is slower than without updating the probabilities (Fig. 5.21a). For PGE, we see that around the 7th mutation ($k = 7$) the growth of the line changes, slowly starting to decrease, and approaching the line of the average between two random individuals. In PSGE we see that with the probabilities' update the line growth is slower, being similar with the growth of GE and PGE until approximately $k = 7$ however it still ends with higher values than the other methods.

In the redundancy analysis, we exposed a theory to explain why GE, PGE and PSGE have higher percentage of mutated individuals with $MI > 100$ with respect to the original individuals. In the locality analysis, only effective mutations are considered ($MI > 0$), so by removing the redundancy from the mutations, the probabilities of generating recursive productions after a mutation is the same between the methods, and therefore we can say that there are more factors contributing to mutations causing many differences between phenotypes.

From the graphs (Fig. 5.21), we saw that GE and PGE have higher mean of MI than SGE, i.e., less locality. This difference had already been studied by Lourenço et al. [24], who argued that the high locality of SGE is a consequence of the representation used by the individuals. In GE and PGE the genotype is a list of codons, and in SGE and PSGE it is a set of dynamic lists, one list for each non-terminal. PSGE uses the same representation as SGE, however the codons are different: in SGE they are the indexes of production rules, while in PSGE they are real values representing the probability of choosing a production rule from the PCFG. Another difference between PSGE and SGE, which may be related to the fact that PSGE has very low locality, is that PSGE uses

a Gaussian mutation. While the other methods use random mutation, i.e. the codon is replaced by a random value, PSGE mutation adds a value generated with a Gaussian distribution to the value of the codon selected for mutation. This difference in the mutation used may be favouring recursive rules, however further research is required to understand the impact of the representation and type of mutation used (different parameters of the Gaussian distribution and different types of mutation such as random should be considered) in terms of performance, redundancy and locality.

This page is intentionally left blank.

Chapter 6

Conclusion

Grammatical Evolution (GE) is a grammar-based Genetic Programming (GP) variant that has attracted the attention of many researchers and practitioners, since its proposal in the late 1990s and it has been applied with success to many problem domains. However, it has been shown that it suffers from some issues, such as low locality and high redundancy [3, 4].

In grammar-based GP, the choice of the grammar has a significant impact on the quality of the generated solutions as it is the grammar that defines the space of possible solutions. Our goal work was to create a variant of GE that could guide the search towards better solutions, inserting bias in the production rules that generate better individuals, in order to achieve a better overall performance, and at the same time overcome the problems of GE.

In this work we proposed four different variants of GE, introducing alternative representations of individuals, a new mapping mechanism relying on a Probabilistic Context-Free Grammar (PCFG), and two techniques to evolve the grammar's probability. In concrete, we present Probabilistic Grammatical Evolution (PGE), in which the genotype of an individual is a variable length sequence of floats, and the genotype-phenotype mapping is performed using a PCFG. Each derivation rule is assigned a probability, which is updated based on the number of times that each rule was expanded by the chosen individual. We alternate between the best individual overall and the best individual of the current generation in order to maintain a balance between global and local exploration. Later on we adapted PGE to Structured Grammatical Evolution (SGE), resulting in Probabilistic Structured Grammatical Evolution (PSGE). The genotype in PSGE is a set of dynamic lists, one for each non-terminal of the grammar. Each codon in the list represents the likelihood of selecting a production rule. The PCFG is updated and individuals are remapped at the end of each generation, same as in PGE.

Co-evolutionary Probabilistic Grammatical Evolution (Co-PGE) uses as representation of individuals a list of probabilities, which are mapped into production rules using a PCFG. Each individual co-evolves with its own grammar, and at each generation the grammar can be mutated, changing the probabilities of the production rules. Co-evolutionary Probabilistic Structured Grammatical Evolution (Co-PSGE) was created by adapting this co-evolutionary process to PSGE. The individuals of Co-PSGE have a set of dynamic lists as their genotype, and each list contains the probabilities that are mapped into production rules for each non-terminal, using the PCFG. At the end of each generation the individuals are remapped.

The proposed methods were compared to the standard versions of GE and SGE on different benchmark problems, analysing the evolution of the mean best fitness. In four of the six problems studied, PGE and Co-PGE were statistically better than GE, while in the remaining two were statistically similar. Despite being better than GE for most problems, the approaches were only able to match SGE's performance in one problem each.

In terms of PSGE and Co-PSGE performance, in all problems they were statistically better than GE. In four cases, PSGE performed statistically similar to SGE, and in two problems, it performed slightly worse. In three instances, the Co-PSGE performed statistically better, in one problem it performed similarly, and in two problems it performed slightly worse.

The results showed that evolving a grammar's probabilities along the evolutionary process can help to guide the solutions towards better fitness, creating an approach that is able to compete with SGE (Co-PSGE).

We were able to validate the usefulness of the evolved PCFG by analysing the evolution of probabilities over generations. We observed a marginal increase in the probability of the non-recursive rule when explosive grammars were used (more recursive than non-recursive production rules). This pattern was observed in all five explosive non-terminals symbols examined. By inserting bias in the minority rule, we are balancing the grammar, which allows us to generate more individuals of varying sizes, which is beneficial for sampling different solutions from the search space [11].

After the evolutionary process, we are left with a grammar specialized for a certain problem, which can be reused to generate a sample population with better initial fitness. In addition, it gives us an idea of which production rules are more relevant to generate better individuals and helps to better understand the characteristics of the problem. On the other hand, in case we have information about the problem, we can also manually adjust the probabilities to guide the search process.

We investigated the methods' locality and found that PGE presents higher locality than GE, which is beneficial for exploitation an exploration of the search space. We also looked at redundancy, and observed that the variants of GE have similar issues of high redundancy. On the other hand, SGE variants showed lower redundancy, being almost zero after 4 consecutive mutations.

6.1 Future Work

We present two different mechanisms to change the bias of grammars. For future work, it would be interesting to explore other approaches.

After the analysis performed, there are some behaviours of the methods that are interesting to explore. Given the results obtained in the locality and redundancy study, future research will be needed to better understand the impact of the representation and variation operators used, and how we can improve the results. We should also conduct an analysis of the distribution of individuals over the search space, and analyse the effect that the parameters (i.e., learning factor, in the case of PGE and PSGE, and mutation probability and standard deviation of Gaussian mutation, in Co-PGE and Co-PSGE) have on the exploration of the solutions. To complete this analysis, it would be interesting to examine the size and shape of the solution trees.

The study carried out on the evolution of probabilities showed us that the methods tend to balance the weights of non-recursive and recursive productions, and introduce bias in

the most important rules to create better solutions. Taking this into account, it will be interesting to examine the response of the algorithms to different types of grammars. One suggestion is to use grammars with different numbers of productions for the same non-terminal, or redundant productions, following, for example, an analysis like the one done by Nicolau et al. [5], in which explosive, balanced, unlinked, with a single non-terminal symbol, and more grammars were used. It will also be interesting to see the behaviour with randomly generated grammars, or grammars biased to generate bad solutions.

References

[1] C. Ryan, M. O'Neill, and J.J. Collins, editors. *Handbook of Grammatical Evolution*. Springer International Publishing, 2018. doi: 10.1007/978-3-319-78717-6. URL https://doi.org/10.1007/978-3-319-78717-6.

[2] J. R. Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4(2), June 1994. doi: 10.1007/bf00175355. URL https://doi.org/10.1007/bf00175355.

[3] F. Rothlauf and M. Oetzel. On the locality of grammatical evolution. In *Lecture Notes in Computer Science*, pages 320–330. Springer Berlin Heidelberg, 2006. doi: 10.1007/11729976_29. URL https://doi.org/10.1007/11729976_29.

[4] F. Rothlauf and D. E. Goldberg. Redundant representations in evolutionary computation. *Evolutionary Computation*, 11(4):381–415, December 2003. doi: 10.1162/106365603322519288. URL https://doi.org/10.1162/106365603322519288.

[5] M. Nicolau and A. Agapitos. Understanding grammatical evolution: Grammar design. In *Handbook of Grammatical Evolution*, pages 23–53. Springer International Publishing, 2018. doi: 10.1007/978-3-319-78717-6_2. URL https://doi.org/10.1007/978-3-319-78717-6_2.

[6] N. Lourenço, F. Assunção, F. B. Pereira, E. Costa, and P. Machado. Structured grammatical evolution: A dynamic approach. In *Handbook of Grammatical Evolution*, pages 137–161. Springer International Publishing, 2018. doi: 10.1007/978-3-319-78717-6_6. URL https://doi.org/10.1007/978-3-319-78717-6_6.

[7] P. A. Whigham and Department Of Computer Science. Grammatically-based genetic programming, 1995.

[8] C. Ryan, J.J. Collins, and M. O Neill. Grammatical evolution: Evolving programs for an arbitrary language. In *Lecture Notes in Computer Science*, pages 83–96. Springer Berlin Heidelberg, 1998. doi: 10.1007/bfb0055930. URL https://doi.org/10.1007/bfb0055930.

[9] P. A. Whigham, G. Dick, J. Maclaurin, and C. A. Owen. Examining the "best of both worlds" of grammatical evolution. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, July 2015. doi: 10.1145/2739480.2754784. URL https://doi.org/10.1145/2739480.2754784.

[10] M. O'Neill and C. Ryan. Grammatical evolution by grammatical evolution: The evolution of grammar and genetic code. In *Lecture Notes in Computer Science*, pages 138–149. Springer Berlin Heidelberg, 2004. doi: 10.1007/978-3-540-24650-3_13. URL https://doi.org/10.1007/978-3-540-24650-3_13.

[11] R. Harper. GE, explosive grammars and the lasting legacy of bad initialisation. In *IEEE Congress on Evolutionary Computation*. IEEE, July 2010. doi: 10.1109/cec.2010.5586336. URL https://doi.org/10.1109/cec.2010.5586336.

[12] M. Nicolau. Automatic grammar complexity reduction in grammatical evolution. 2004.

[13] M. O'Neill, A. Brabazon, M. Nicolau, S. McGarraghy, and P. Keenan. grammatical evolution. In *Genetic and Evolutionary Computation – GECCO 2004*, pages 617–629. Springer Berlin Heidelberg, 2004. doi: 10.1007/978-3-540-24855-2_70. URL https://doi.org/10.1007/978-3-540-24855-2_70.

[14] H. Kim and C. W. Ahn. A new grammatical evolution based on probabilistic context-free grammar. In *Proceedings in Adaptation, Learning and Optimization*, pages 1–12. Springer International Publishing, 2015. doi: 10.1007/978-3-319-13356-0_1. URL https://doi.org/10.1007/978-3-319-13356-0_1.

[15] H. Kim, H. Kang, and C. W. Ahn. A conditional dependency based probabilistic model building grammatical evolution. *IEICE Transactions on Information and Systems*, E99.D(7):1937–1940, 2016. doi: 10.1587/transinf.2016edl8004. URL https://doi.org/10.1587/transinf.2016edl8004.

[16] N. Lourenço, F. B. Pereira, and E. Costa. SGE: A structured representation for grammatical evolution. In *Lecture Notes in Computer Science*, pages 136–148. Springer International Publishing, 2016. doi: 10.1007/978-3-319-31471-6_11. URL https://doi.org/10.1007/978-3-319-31471-6_11.

[17] C. Ryan, A. Azad, A. Sheahan, and M. O'Neill. No coercion and no prohibition, a position independent encoding scheme for evolutionary algorithms – the chorus system. In *Lecture Notes in Computer Science*, pages 131–141. Springer Berlin Heidelberg, 2002. doi: 10.1007/3-540-45984-7_13. URL https://doi.org/10.1007/3-540-45984-7_13.

[18] J. Mégane, N. Lourenço, and P. Machado. Probabilistic grammatical evolution. In Ting Hu, Nuno Lourenço, and Eric Medvet, editors, *Genetic Programming*, pages 198–213, Cham, 2021. Springer International Publishing. ISBN 978-3-030-72812-0.

[19] M. Nicolau. Understanding grammatical evolution: initialisation. *Genetic Programming and Evolvable Machines*, 18(4):467–507, July 2017. doi: 10.1007/s10710-017-9309-9. URL https://doi.org/10.1007/s10710-017-9309-9.

[20] C. Ryan and R. M. A. Azad. Sensible initialisation in grammatical evolution. In Alwyn M. Barry, editor, *GECCO 2003: Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference*, pages 142–145, Chigaco, 11 July 2003. AAAI.

[21] D. Fagan, M. Fenton, and M. O'Neill. Exploring position independent initialisation in grammatical evolution. In *2016 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, July 2016. doi: 10.1109/cec.2016.7748331. URL https://doi.org/10.1109/cec.2016.7748331.

[22] S. Luke. Two fast tree-creation algorithms for genetic programming. *IEEE Transactions on Evolutionary Computation*, 4(3):274–283, 2000. doi: 10.1109/4235.873237. URL https://doi.org/10.1109/4235.873237.

[23] E. Murphy, E. Hemberg, M. Nicolau, M. O'Neill, and A. Brabazon. Grammar bias and initialisation in grammar based genetic programming. In *Lecture Notes in Computer Science*, pages 85–96. Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-29139-5_8. URL https://doi.org/10.1007/978-3-642-29139-5_8.

[24] N. Lourenço, F. B. Pereira, and E. Costa. Unveiling the properties of structured grammatical evolution. *Genetic Programming and Evolvable Machines*, 17(3):251–289, February 2016. doi: 10.1007/s10710-015-9262-4. URL https://doi.org/10.1007/s10710-015-9262-4.

[25] E. Medvet. A comparative analysis of dynamic locality and redundancy in grammatical evolution. In *Lecture Notes in Computer Science*, pages 326–342. Springer International Publishing, 2017. doi: 10.1007/978-3-319-55696-3_21. URL https://doi.org/10.1007/978-3-319-55696-3_21.

[26] G. R. Raidl and J. Gottlieb. Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem. *Evolutionary Computation*, 13(4):441–475, December 2005. doi: 10.1162/106365605774666886. URL https://doi.org/10.1162/106365605774666886.

[27] C. Darwin. *On the Origin of Species by Means of Natural Selection*. Murray, London, 1859. or the Preservation of Favored Races in the Struggle for Life.

[28] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer Berlin Heidelberg, 2015. doi: 10.1007/978-3-662-44874-8. URL https://doi.org/10.1007/978-3-662-44874-8.

[29] R. Poli, W. B. Langdon, and N. F. McPhee. *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, 2008. ISBN 1409200736.

[30] E. Burke, S. Gustafson, and G. Kendall. Ramped half-n-half initialisation bias in GP. In *Genetic and Evolutionary Computation — GECCO 2003*, pages 1800–1801. Springer Berlin Heidelberg, 2003. doi: 10.1007/3-540-45110-2_71. URL https://doi.org/10.1007/3-540-45110-2_71.

[31] J. R. Woodward. Modularity in genetic programming. In *Lecture Notes in Computer Science*, pages 254–263. Springer Berlin Heidelberg, 2003. doi: 10.1007/3-540-36599-0_23. URL https://doi.org/10.1007/3-540-36599-0_23.

[32] G. Gerules and C. Janikow. A survey of modularity in genetic programming. *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 5034–5043, 2016.

[33] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge, MA, USA, 1994. ISBN 0262111896.

[34] M. O'Neill and C. Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Springer US, 2003. doi: 10.1007/978-1-4615-0447-4. URL https://doi.org/10.1007/978-1-4615-0447-4.

[35] A. Bartoli, M. Castelli, and E. Medvet. Weighted hierarchical grammatical evolution. *IEEE Transactions on Cybernetics*, 50(2):476–488, November 2018. doi: 10.1109/tcyb.2018.2876563. URL https://doi.org/10.1109/tcyb.2018.2876563.

[36] D. Fagan, M. O'Neill, E. Galván-López, A. Brabazon, and S. McGarraghy. An analysis of genotype-phenotype maps in grammatical evolution. In *Lecture Notes in Computer Science*, pages 62–73. Springer Berlin Heidelberg, 2010. doi: 10.1007/978-3-642-12148-7_6. URL https://doi.org/10.1007/978-3-642-12148-7_6.

[37] N. Lourenço, J. Ferrer, F. B. Pereira, and E. Costa. A comparative study of different grammar-based genetic programming approaches. In *Lecture Notes in Computer Science*, pages 311–325. Springer International Publishing, 2017. doi: 10.1007/978-3-319-55696-3_20. URL https://doi.org/10.1007/978-3-319-55696-3_20.

[38] K. Kim, Y. Shan, N. X. Hoai, and R. I. McKay. Probabilistic model building in genetic programming: a critical review. *Genetic Programming and Evolvable Machines*, 15 (2):115–167, September 2013. doi: 10.1007/s10710-013-9205-x. URL https://doi.org/10.1007/s10710-013-9205-x.

[39] P. Larrañaga and J. A. Lozano, editors. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Springer US, 2002. doi: 10.1007/978-1-4615-1539-5. URL https://doi.org/10.1007/978-1-4615-1539-5.

[40] M. O'Neill and A. Brabazon. mGGA: The meta-grammar genetic algorithm. In *Lecture Notes in Computer Science*, pages 311–320. Springer Berlin Heidelberg, 2005. doi: 10.1007/978-3-540-31989-4_28. URL https://doi.org/10.1007/978-3-540-31989-4_28.

[41] A. Joshi. Tree-adjoining grammars and lexicalized grammars ms-cis-91-22 linc lab 197. 1991.

[42] E. Anders and P. Hemberg. An exploration of grammars in grammatical evolution. 2010.

[43] J. McDermott, K. De Jong, U. O'Reilly, D. R. White, S. Luke, L. Manzoni, M. Castelli, L. Vanneschi, W. Jaskowski, K. Krawiec, and R. Harper. Genetic programming needs better benchmarks. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference - GECCO '12*. ACM Press, 2012. doi: 10.1145/2330163.2330273. URL https://doi.org/10.1145/2330163.2330273.

[44] T. Bäck, G. Rudolph, and H. Schwefel. Evolutionary programming and evolution strategies: Similarities and differences. In *In Proceedings of the Second Annual Conference on Evolutionary Programming*, pages 11–22, 1997.

[45] R. Hinterding. Gaussian mutation and self-adaption for numeric genetic algorithms. In *Proceedings of 1995 IEEE International Conference on Evolutionary Computation*, volume 1, pages 384–, 1995. doi: 10.1109/ICEC.1995.489178.

[46] H. Beyer and H. Schwefel. Evolution strategies – a comprehensive introduction. *Natural Computing*, 1:3–52, 2004.

[47] D. Harrison and D. Rubinfeld. Boston Housing Data. http://lib.stat.cmu.edu/datasets/boston, 1993. [Online; accessed 27-December-2020].

[48] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. 1(1): 67–82. doi: 10.1109/4235.585893.

[49] J. Che, Y. Yang, L. Li, X. Bai, S. Zhang, and C. Deng. Maximum relevance minimum common redundancy feature selection for nonlinear data. *Information Sciences*, 409-410:68–86, October 2017. doi: 10.1016/j.ins.2017.05.013. URL https://doi.org/10.1016/j.ins.2017.05.013.

[50] W. B. Langdon and R. Poli. Why ants are hard. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998:*

Proceedings of the Third Annual Conference, pages 193–201, University of Wisconsin, Madison, Wisconsin, USA, 22-25July 1998. Morgan Kaufmann. ISBN 1-55860-548-7. URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/WBL.antspace_gp98.pdf.

[51] D. Wilson and D. Kaur. How santa fe ants evolve. *arXiv: Neural and Evolutionary Computing*, 2013.

[52] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262, December 1989. doi: 10.1137/0218082. URL https://doi.org/10.1137/0218082.