Grammar Based Evolutionary Design

· U  C ·

Filipe Guerreiro Assunção
fga@student.dei.uc.pt

Faculty of Sciences and Technology
University of Coimbra

Advisors

Dr. Penousal Machado
Dr. James McDermott

Coimbra 2015

Examiners

Dr. Luis Miguel M. L. Macedo
Dr. Marco P. Vieira

Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning.

Winston Churchill

Acknowledgements

I would like to acknowledge the contribution of the Computational Design & Visualization Lab. (University of Coimbra, Portugal) and of the Natural Computing Research & Applications Group (University College Dublin, Ireland) both for mentoring, as well as for technical support. I would specially like to thank to my advisors, Dr. Penousal Machado and Dr. James McDermott. Without you, none of the work herein described would be possible.

To my parents, girlfriend and friends all my gratitude for listening to my frustrations when things were going mad. For dragging me from the computer when I was hopelessly trying to fix code or just stuck writing some part of this document. Those breaks were exactly what I needed to keep going.

Finally, I would like to acknowledge project ConCreTe, for the provided funding. The project ConCreTe acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET grant number 611733.

Agradecimentos

Gostaria de agradecer toda a contribuição do *Computational Design & Visualization Lab.* (Universidade de Coimbra, Portugal) e do *Natural Computing Research & Applications Group* (University College Dublin, Irlanda) por toda a ajuda tanto a nível científico, como técnico. Agradecimentos especiais aos meus orientadores, Dr. Penousal Machado e Dr. James McDermott. Sem vocês, a realização de todo trabalho aqui descrito seria impossível.

Aos meus pais, namorada e amigos, toda a minha gratidão por ouvirem as minhas frustrações quando algo não corria de acordo com o planeado. Por me arrastarem para longe do computador quando, desesperadamente, tentava corrigir código ou simplesmente estava preso em alguma parte da escrita. Essas pausas eram exatamente o que precisava para conseguir continuar.

Finalmente, gostaria de reconhecer o projeto ConCreTe, por todo o financiamento que me foi disponibilizado.

Abstract

A novel evolutionary approach for evolving grammars is presented. It is composed of two main methodologies. In the first one, individuals are represented as a graph, where nodes stand for production rules, and connections between them to the flow of control and parameters. In the second one, a tree representation is used; individuals are derivation trees, from a pre-defined grammar. Inner nodes represent non-terminals, and leaves terminals. It is proved that both representations are able to properly evolve the desired solutions (in this case, images and music).

After conducting experiments with the above approaches, running separately, we try to merge components from both of them in order to assess if any improvement is achieved. Graph-mutation has proven to generate the worst results, using both tree and graph-crossovers. When tree-mutation is used the performance obtained with tree and graph-crossovers tends to be similar. Nevertheless, an analysis at a population level, reveals that tree-crossover outperforms the graph one.

Considering these results, we searched forms of improving the graph-crossover operator, by means of alignment, thus taking into account the structure of the graphs. Aligning two graphs choosing then the cutting points from the list returned by the algorithm, where the matched pairs of nodes have the least alignment cost, has proven to lead to better performance.

Finally, we address the problem of assessing the quality of families of images, proposing a fitness function for the assessment of families of individuals. Families are seen as sets of artifacts that should share common characteristics, allowing one to intuitively classify them as belonging to the same family. Results show the validity of the method and prove that, to evolve a family, both the qualities of the set and of each individual must be taken into consideration.

Keywords. Genetic Programming, Grammar Evolution, Alignment, Fitness Assignment

Resumo

Apresentamos uma nova abordagem evolucionária para a evolução de gramáticas. É composta por duas metodologias principais. Na primeira, os indivíduos são representados por grafos, onde os nós internos codificam regras de produção e as ligações entre eles denotam controlo de fluxo e parâmetros. Na segunda metodologia, usamos uma representação em árvore. Nós internos simbolizam símbolos não terminais, enquanto que as folhas representam símbolos terminais. É provado que ambas as metodologias são capazes de corretamente evoluir as soluções desejadas (neste caso, imagens e música).

Após a realização de testes com as abordagens anteriormente mencionadas, em separado, efetuamos novas experiências juntando os operadores de cada uma delas, com o objetivo de investigar se tal conduz a melhorias de desempenho. É demonstrado que a mutação de grafos leva aos piores resultados, tanto utilizando o cruzamento de grafos como o de árvores. Quando utilizada a mutação de árvores, com uma ou outra forma de cruzamento, os resultados são semelhantes. No entanto, uma análise ao nível populacional revela que o cruzamento de grafos gera resultados considerados superiores aos obtidos pelo cruzamento de árvores.

Tendo em conta os resultados anteriores, formas de melhorar o cruzamento de grafos foram investigadas, recorrendo a técnicas de alinhamento e, como tal, considerando a estrutura dos grafos. Alinhar dois grafos, escolhendo depois os pontos de corte com base na lista retornada pelo algoritmo, onde os pares de nós correspondidos tem um baixo custo de alinhamento, provou ser capaz de conduzir a uma melhoria de desempenho.

Por fim, focamos o problema de atribuição de qualidade a famílias de imagens, propondo para tal uma função capaz de classificar a qualidade de uma família de indivíduos. Uma família é considerada um conjunto de artefatos, que devem possuir características comuns, permitindo identificá-los como pertences à mesma família. Os resultados provam que, se considerada a qualidade da família como um todo, bem como a qualidade individual de cada um dos indivíduos que a compõe, é possível evoluir conjuntos passíveis de serem identificados como famílias.

Palavras Chave. Programação Genética, Evolução de Gramáticas, Alinhamento, Atribuição de Fitness

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Grammars

List of Matrices

Chapter 1

Introduction

In 1859, Darwin published a book entitled *On the Origin of Species* [13] that would forever change the perceptions about the evolution of individuals across generations, i.e., from offspring to offspring. It was in this book that the principles and evidences of natural selection were first presented. This theory focuses on the fact that, in a population of individuals, they will compete to survive and, therefore, only the ones most suit to the environmental characteristics would be fit enough to breed, producing the new offspring. This principle, also known as survival of the fittest is repeated in time, leading to the evolution of the species.

Later, the above mentioned theory was adapted to Artificial Intelligence, which can be perceived as systems able to take decisions with the information they sense from the environment. One of the first algorithms proposed using the natural evolution theory of Darwin was Genetic Algorithms, introduced by Holland in the 1960s [27]; simplistically, it works as a search heuristic. In this method, a set of candidate solutions (population in nature) is evolved where, from generation to generation, the best individuals (those best suit to the environmental characteristics) are breed in order to form the new offspring.

Several other approaches based on the same theory were proposed. The one we are going to explore during the course of the present Dissertation is Genetic Programming (GP). In this type of technique individuals stand for complete and executable computer programs.

Considering that several domains can be represented using grammatical formulations, we aim at developing a tool capable of evolving solutions in multiple environments. As such, we are going to research and study, in the next chapters, ways of doing so by combining GP with grammar formulations.

In the following sections we will start by defining the scope and goals of this Dissertation (Sections 1.1 and 1.2, respectively). Then, we will introduce document

assumptions (Section 1.3) and, to end, in Section 1.4, document structure is going to be described.

1.1 Scope

As mentioned earlier, we will focus on works capable of manipulating and evolving grammars, where individuals are generated from a pre-defined grammar formulation. In other words, individuals are themselves derivations of a higher-level grammar.

After revising the state of the art about the above subjects, we will propose approaches that combine GP with grammar theory, as a form of tackling the problem of generalisation, i.e., we want to develop a method to evolve individuals resembling grammars, but it cannot be domain specific.

Notwithstanding the range of application of the work that will be developed and detailed during the course of this Dissertation, it is our intent to apply it mainly over the evolution of 2D images (generated by the evolution of Context Free Design Grammars [11]). However, in a later stage, we will also test it with other grammatical scenarios, namely music, to confirm if it has sufficient generalisation capacities to be able to produce good quality solutions, under different circumstances.

Focus is given to both these domains because of the requirements of the European Funded Project ConCreTe[1], where the current Dissertation is placed.

1.2 Goals

The main goal of the current Dissertation is the proposal of a novel approach capable of evolving grammars that follow a specific formulation, which is a-priori defined by the user. By doing so, the system will bear an important property: it would be applicable to any domain where a grammar can be used to describe it.

Building on previous research [41, 42], we will study two different approaches. In the first one, individuals are represented as graphs, where the whole set of nodes and connections between them represents the production rules of a grammar. Later on, a method using derivation trees will be also inspected, following a more traditional form of GP.

While developing both algorithms, we faced some question that would be interesting to address, such as the generation of families of individuals. We decided to tackle then, which is easily seen from the remainder of the document.

[1]For more informations check http://conceptcreationtechnology.eu/

1.3 Assumptions

During the development of the present document, multiple times there is the need to perform statistical tests over collected data. All statistical validations were conducted under a 95% confidence level. Additionally, as data never follows a normal distribution, and as the initial conditions of the experiments are always the same, if more than two categories are under comparison we use the Friedmans's ANOVA statistical test. If there is proven that the results are statistically different the Wilcoxon test is then applied to compare all possible pairs. If just two categories are in comparison only the last test is used.

1.4 Document Structure

The remainder of the document is organised as follows. Chapter 2 starts by presenting a short overview of Evolutionary Computation, focusing then on Genetic Programming and Grammar Evolution, which are the main topics of this Dissertation and are described as the central point for the creation of the evolutionary engine approaches herein presented, in Chapter 3.

Chapter 4 details the experiments performed over the methodologies described in Chapter 3. Tests are conducted over different domains and setups, focusing distinct aspects of the evolutionary engine.

After finishing the above experiments we noticed that there was still a possibility to increase the performance of the operators introduced in Chapter 3. For that, we explored alignment techniques. Both technical description and experiments regarding alignment of structures can be found in Chapter 5.

In Chapter 6 we present a fitness function which aims at assessing the quality of a set of individuals, taking into account the set as a whole instead of just the individual quality of its members. Considering that, we evolve families, making use of the non-deterministic nature of some forms of grammars, i.e., when they are mapped multiple times from their grammatical representation into a concrete output (image / sound / etc.) different results can be generated.

To end, conclusions regarding all the work developed during the course of this Dissertation are drawn and future work is addressed (Chapter 7).

Chapter 2

State of the Art

This chapter aims at describing and presenting the previous work that has been done in the fields within the scope of this Dissertation. From that, it is easily understood that most of the time will be spent analysing Evolutionary Computation (EC) works from two different, yet, important perspectives. To start, focus will be given to Genetic Programming, moving then to EC techniques that have been resorting to grammatical ways of guiding evolution.

Despite not being the focus of this Dissertation but, as much of the developed work will be tested under the evolution of artworks, some research was also conducted in the field of Evolutionary Art. Because this is just a field of application and not the main focus of this document its content is presented in Appendix A.

The organisation of this Chapter is as follows. In Section 2.1 an introduction to EC, detailing the principles that guide it, is provided. An overview of what are Evolutionary Algorithms (EAs) is presented in Section 2.2, introducing the different types of EA approaches. This last section is divided into two subsections; the first one (Section 2.2.1) details the components shared by all EAs, whereas the latter one (Section 2.2.2) focuses specifically on Genetic Programming (featuring different approaches to it). In Section 2.3 we start by giving a formal introduction to grammars and then, in the following subsections (2.3.1 and 2.3.2), Genetic Programming techniques capable of dealing with their evolution are presented.

2.1 Evolutionary Computation

Evolutionary Computation (EC) [2, 18, 72] is a field of research within Computer Science, related to the Darwinian natural evolution of species [13].

In this theory, a population of individuals, subjected to a given environment, will struggle for survival and chance to reproduce, due to the fact that the environment

can only host a limited number of individuals. Higher quality individuals, those that are best suited to the environment characteristics, will have a higher probability of surviving and reproduction, passing their characteristics to the offspring.

This can all be translated into the principle of survival of the fittest, i.e., if individuals have characteristics that make them well adapted to the environment they are propagated to the offspring. Else, they are discarded by dying without children. Occasionally, changes in the genetic material (mutations) can happen, altering the constitution of the population. All this together is what makes evolution possible.

The previously described natural evolutionary process can be intuitively adapted to computation, mainly over generate-and-test problems (also known as trial-and-error problems). In this type of problem it is possible to generate a possible solution and evaluate its quality in some way, promoting evolution.

It becomes now clear that a mapping between natural evolution and problem solving can be established. The environment in natural evolution corresponds to the problem in problem solving, the individual becomes the candidate solution and the fitness the quality of the candidate solutions.

As problems increase their complexity and machines still have a limited capacity, it becomes infeasible to search for optimal solutions. This justifies the need of this kind of heuristic approaches, which are mainly used when a good solution, not necessarily the optimal one, is sought within an acceptable time.

In particular, there are three types of problems that normally resort to EC:

Optimisation – when it is needed to find the inputs that lead to a known optimal output, given the model;

Modelling / System Identification – both the inputs and outputs are known, although it is needed to find out what is the model that maps the inputs into the outputs;

Simulation – the model and some inputs are known. There is the need to find the outputs for the corresponding inputs.

2.2 Evolutionary Algorithms

Evolutionary Algorithms (EAs) [10,18] are a subset of EC and, as such, they also follow the natural evolution theory of Darwin. Multiple types of EAs exist: Evolutionary Programming (EP) [20], Evolution Strategies (ES) [6], Genetic Algorithms (GA) [27] and Genetic Programming (GP) [32]. They all share the same core components.

To begin, an initial population of individuals is needed. This set of candidate solutions will be evolved over time, subjected to environmental pressure, leading to higher quality solutions.

The whole process of evolution is guided by a function (normally referred to as fitness function) that must be capable of measuring a candidate's solution quality. If solving a maximisation problem, the higher the fitness value is the better. In the other hand, if trying to solve a minimisation problem, the lower the value the better.

Variation operators are also applied to candidate solutions. Recombination is a well known operator that aims at generating one or more new candidates (children), based on the exchange of genetic material between two or more individuals (parents). The selection of the parents is a process based on the fitness value of the candidates, where commonly the probability of choosing an individual is a function of its quality. This way, all individuals have a probability, even if low, of being chosen. The other used operator is mutation; it only applies changes in the genotype of a selected individual, creating a new one.

Mutation and crossover together achieve variation, a necessary component of search. By making small variations they carry out exploitation. By making larger variations they carry out exploration. Both operators can achieve small and large variations in different settings. Exploration is also linked with the principle of global search and exploitation with local search. Exploitation and exploration must work together to achieve successful search, promoting diversity and thereby facilitating novelty.

Upon application of these operators, and as population size is typically fixed, it is necessary to choose which individuals form the offspring. Usually, they are chosen taking into account one of the following criteria: fitness or age.

Algorithm 1 Typical Evolutionary Algorithm [18].

Initialise population with random candidate solutions;
Evaluate each candidate;
while termination condition is not satisfied **do**
    1 - Select parents;
    2 - Recombine pairs of parents;
    3 - Mutate the resulting offspring;
    4 - Evaluate new candidates;
    5 - Select individuals for the next generation.
end while

All the steps above mentioned are then repeated until a stop condition is met,

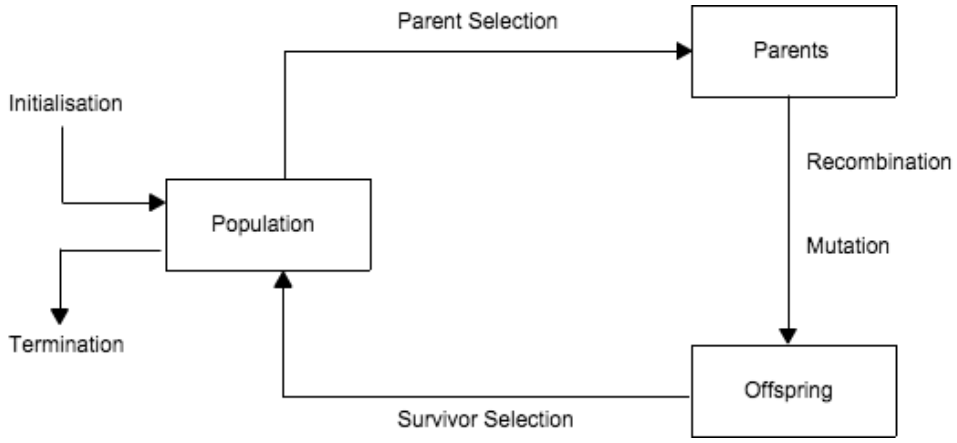


Figure 2.1: Typical Evolutionary Algorithm flow-chart [18].

	GA	ES	EP	GP
Typical problems	Combinatorial optimisation	Continuous optimisation	Optimisation	Modelling
Typical representation	Strings over a finite alphabet	Strings (vectors) of real numbers	Application specific often as in ES	Trees
Role of recombination	Primary variation operator	Important, but secondary	Never applied	Primary / only variation operator
Role of mutation	Secondary variation operator	Important, sometimes the only operator	The only variation operator	Secondary, sometimes not used at all
Parent selection	Random, biased by fitness	Random, uniform	Each individual creates one child	Random, biased by fitness
Survivor selection	Generational: n.a. all individuals replaced Steady-state: deterministic biased by fitness	Deterministic, biased by fitness	Random, biased by fitness	Random, biased by fitness

Table 2.1: Differences between EA approaches [18].

promoting the continuous evolution of populations, one after the other. A systematic overview of the textual description can be seen as a flow diagram, in Figure 2.1 and as pseudo-code, in Algorithm 1.

As stated at the beginning of the current section, there are several types of Evolutionary Algorithms. They all follow the previously referred steps; although, they have differences regarding the kind of problems they aim at solving and in the details of its implementation. One of the aspects that varies most from approach to approach is the representation. This has several implications in the implementation of the majority of the rest of the operators as they are dependent from it. The main differences between the various EAs are briefly presented in Table 2.1.

2.2.1 Components of Evolutionary Algorithms

In this section the components that together make an Evolutionary Algorithm are going to be discussed and presented in a much deeper level of detail. As previously seen, in Figure 2.1, to build an EA the following parts are needed:

- Representation of individuals;
- Population;
- Population initialisation;
- Evaluation of candidate solutions;
- Parent selection;
- Variation operators;
- Survivors selection;
- Termination condition.

2.2.1.1 Representation

Representation is one of the most important components of an EA, because it has direct impact in several other parts, namely in recombination and mutation. In other words, we can say that the variation operators are representation dependent, meaning that they must be adapted to it.

A rule of thumb is to choose as representation model the one that matches the given problem better, i.e., the one that makes the encoding of individuals easier or more natural.

This component is of huge importance because it is here that the mapping between the original problem context (phenotype) and the domain of the candidate solutions of the problem (genotype) is accomplished. As in nature, the phenotype is the expression of the individual's physic characteristics, whereas the genotype is the set of genes[1] that encodes the phenotype.

Encoding and decoding reveal an import property of representation: it has to be invertible; there has to be one genotype that corresponds to a given phenotype and a phenotype that corresponds to a genotype.

Examples of standard representations are binary or real-valued vectors, trees or graphs [69].

[1]Genetic unit that encodes a property of the individual.

2.2.1.2 Evaluation Function

An evaluation function, commonly referred to as fitness function, quality function or cost function, is what allows EAs to classify and compare different candidate solutions. In other words, it is the definition of improvement and works as a representation of the task to be solved.

At a more detailed level, the fitness function ranks the genotype with reference to its phenotipic capacity of solving the problem under study. For example, if maximising, better suited solutions must have higher fitness values, and the opposite; when minimising, better suited solutions must have lower fitness values.

In some problems, the representation allows invalid individuals; they must be penalised. This is done by the evaluation function, that should rank those invalid solutions with lower fitness values. In this cases, the fitness function can be seen as the normal value that would be attributed to the genotype, minus a penalisation value, that can be based on a linear, quadratic or logarithmic method. Another way of dealing with invalid candidate solutions is to fix them before evaluation.

2.2.1.3 Population

In EAs the population is referred to as the unit of evolution, i.e., it is a set formed by candidate solutions that is evolved over time with the guidance of a fitness function. The only operator that acts directly over the population as a whole is survivors selection (further detailed in Section 2.2.1.6).

Normally, when working with standard EAs, the population size is constant over time. This generates a question: how big should the population be? If, in one hand, bigger populations allow a higher diversity (number of different candidate solutions), reducing the risk of convergence to local optima[2], it is also true that more candidate solutions require longer evaluation time. Another approach is to go for an adaptive population size, as mentioned in [17]. Usually, in this type of methods, population size varies in relation with diversity.

Diversity is highly linked with two concepts previously presented: phenotype and genotype. It can be defined as a measure of rating how many different solutions are present in the population. This may be accomplished by counting distinct genotypes, phenotypes or fitness values. Although, it is important to bear in mind that the same value of fitness can correspond to two genetically different individuals, but not

[2]Solution that is optimal within a region of the domain, although is not the best of all existent optima.

vice-versa. In [52] a method for measuring the diversity of a population taking into consideration both phenotypes and genotypes is proposed.

2.2.1.4 Parent Selection

As previously mentioned, in order to evolve, an EA must choose individuals (parents) that when combined produce the offspring. Parent selection has the main goal of being capable of distinguishing which individuals, when bred, may lead to better solutions. This decision is taken based on the quality of the population individuals, allowing higher-fitness members (those better adapted to the environment) to become parents.

Even aiming at the best possible solution, this operator cannot choose only the best individuals as parents. If so, it will risk a loss in diversity and consequent stagnation, failing to search areas of the problem's domain. For that reason, despite the fact that higher quality individuals have better chances of being chosen, low quality individuals are also given an opportunity to become parents. This relation between the probability of choosing higher and lower fitted individuals is known as selection pressure. High selection pressure implies less diversity, consequently leading to a greater probability of getting stuck in a local optimum. Very low selection pressure approximates random search and, as such, slower evolution.

Amongst the most usual parent selection methods are: roulette wheel, stochastic universal sampling and tournament selection. The ones used in the present Dissertation are now described.

Roulette Selection Also known as fitness proportionate selection. In simple words, if the fitness of an individual, x_i, is given by function f then, the probability of choosing this individual ($prob(x_i)$) is given by Equation 2.1, where N represents the population size. It is now possible to pick p parents by generating p random numbers, between 0 and 1, which will then be mapped to a specific individual, according to their probabilities.

$$prob(x_i) = \frac{f(x_i)}{\sum_{j=1}^{N} f(x_j)} \tag{2.1}$$

Tournament Selection In this method, to select an individual, it is necessary to specify the tournament size (t_{size}), that mentions how many candidate solutions will participate in this "competition". After picking, completely randomly and unbiased, t_{size} individuals from the population, the one with the highest fitness is selected to become a parent. This process is repeated as many times as the number of pretended parents. The choice of the t_{size} parameter is of high importance in what concerns the selection pressure. Higher values of t_{size} give higher selection pressure and vice-versa.

2.2.1.5 Variation Operators

Variation operators, namely recombination and mutation, are responsible for introducing new individuals and diversity in the population, by generating offspring. For that, they use candidate solutions, picked from parent selection methods, and apply to them stochastic changes aiming at producing new genetic material. A deeper explanation of these operators follows.

Recombination selects two or more individuals (parents) - normally two - and recombines their genetic material giving origin to individuals that should express qualities from both parents (refinement of candidate solutions). Preferentially, higher fitted candidate solutions are chosen. It is stochastic due to the choice that is made about what parts of each parent are combined in order to form the offspring.

Mutation is capable of generating a new individual by changing one or more genes of the given candidate solution. The input of mutation is called parent and the output child or offspring. Can be also defined as a stochastic method aimed at producing random unbiased change. It is mainly important in the evolutionary process for providing solutions capable of searching other areas of the problem's domain, avoiding stagnation and convergence to local optima.

Both operators are needed to pursue the objective of finding the optimal solution [73]. Whereas recombination aims preferentially at a local search (exploitation) by combining two higher fitted individuals, hoping to refine the search, mutation goes after new areas of the domain, that may have not been explored yet, assuring a global search (exploration). Other aspect regards the probability of occurrence of each operator. They can be a-priori defined, which is not always easy or an adaptive method can be used [76].

2.2.1.6 Survivor Selection

Survival selection mechanisms, also known as replacement, have the goal to choose which candidate solutions pass from one generation to the other, given that, at the end of each generation the set of candidate solutions is composed by the parents and their offspring.

As already mentioned, normally, in EC, the population size is kept the same during the evolutionary process. So, in order to choose which individuals are passed to the next generation, a process of selection biased by their fitness value is used. Another possibility is to take this decision with regard to their age (parents vs. children).

The most used techniques are to rank all the parents and offspring and select a percentage of the best ones (fitness biased) or to select just the offspring (age biased). Although, another approach is commonly used: elitism. When selecting as survivors only the offspring (generational selection), there is a probability of losing the best individuals. To avoid that, elitism is applied and it consists of passing to the next generation a percentage, normally low, of the best candidate solutions, avoiding this way the deterioration of quality of the population.

When comparing with parent selection an important difference has to be pointed out. While parent selection is known to be a stochastic process, replacement is often deterministic. Being based on rankings or age, the produced results happen to be always the same.

2.2.1.7 Initialisation

Initialisation is the process of creating the first set of candidate solutions that will form the initial generation. To create this individuals it is first important to know whether or not a-priori knowledge about the problem exists. If so, it should be considered in the creation of individuals. Otherwise, they should be created in a randomised way, assuring the covering of most of the domain, to avoid premature convergence and stagnation.

2.2.1.8 Termination Condition

Termination condition is what defines when an evolutionary algorithm should stop evolving, that is, stop trying to find new candidate solutions better than the previous ones. If the optimal value is known, the most used stop criteria is its reaching, or achieving a candidate solution which fitness is within a defined margin of error. But, as the evolution of EAs is randomly determined it is not possible to assure that it is

going to reach the optimal solution. For that, other termination criteria have to be defined. The most usual ones are:

- Definition of the maximum number of fitness evaluations, which indirectly sets the maximum number of generations;
- Diversity between population individuals inferior to a defined threshold;
- Fitness improvement under a threshold for a given period of time (this period of time could be, for example, a number of generations);
- Definition of a maximum allowed CPU execution time.

Instead of using just as stop criteria the convergence to an optimal value, within an acceptable error margin, and because EAs are stochastic, meaning they do not offer guarantees of getting close to the optimal solution, the possibility of combining this condition with one of the other points presented above should be considered. This assures that the algorithm will output solutions within an acceptable period of time.

2.2.2 Genetic Programming

Genetic Programming (GP) [10, 12, 18, 32, 65] is a branch of Evolutionary Algorithms which aims at automatic resolution of complex problems, requiring few information about the form or structure of the solution.

All problems where the solution can be modelled, in any language, as a computer program and where it is possible to define a metric of evaluation, capable of comparing two solutions assessing which one is better are possible to solve using GP.

At a higher level, GPs objective is to evolve executable programs (through the natural process of evolution) that provide the desired solution to a specific problem. This can be seen as one of the main differences between GP and other EAs (GA, ES and EP) because, while the others evolve solutions that optimise a given problem, GP is capable of evolving models (programs) that, when executed, lead to the desired solutions. For that, it is just needed to specify what needs to be done, with no concerns about how it should be accomplished. Other differences exist; while in most EAs individuals are normally represented by linear vectors of fixed size, in GP they are non-linear structures of variable size.

As in any Evolutionary Algorithm, in GP, a population of candidate solutions is evolved over time, being each candidate solution, as previously mentioned, a computer

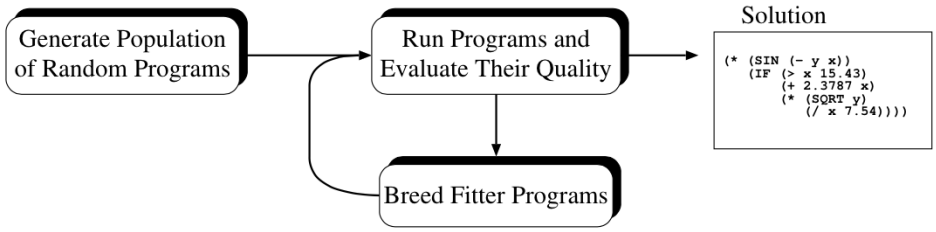


Figure 2.2: Genetic Programming flow-chart [65].

program. From generation to generation, transformations are applied to individuals (namely mutation and crossover), generating offspring. Then, the programs that carry on to the next generation have to be chosen using any of the well known survivor selection mechanisms. A step that is slightly different from the standard EAs is the assessment of quality of the candidate solutions. While in GA, ES and EP fitness is inferred directly from the individual's genotype here, the program has to be executed and the ouputs compared with the expected ones. The flow described in this paragraph is depicted, briefly, in Figure 2.2.

Genetic Programming has been applied with success to a wide range of areas, such as, pattern recognition, symbolic regression, robotics, game strategies, image processing, art, ...

Several types of GP branches exist, with their differences mainly at the level of representation. Although, as mentioned in several other sections, changing the representation has direct impact in several other components of the algorithm, as they are representation dependent. The ones that are going to be explored, in the following subsections are Tree-Based Genetic Programming (Section 2.2.2.1) and two types of Graph-Based Genetic Programming (Sections 2.2.2.2 and 2.2.2.3).

Tree-Based GP is going to be presented because it is the most widely used form of GP and Graph-Based GP will be explored due to the fact that it is also going to be used to develop the evolutionary engine of this Dissertation.

2.2.2.1 Tree-Based Genetic Programming

Proposed by Koza in 1992, it aims, as any GP technique, at solving a problem by developing an executable program that represents the solution.

In the following subsections the details of this approach are going to be presented, so that it becomes clear how the principles of EAs components were adapted to cope with a tree representation.

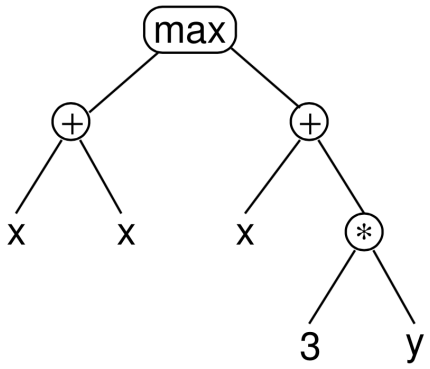


Figure 2.3: GP syntax tree for the program $max(x + x, x + 3 * y)$ [65].

Population, parent and survivor selection and termination conditions are not going to be referred for the fact that no significant differences exist.

Representation This form of GP has its representation based on a tree hierarchy that should capture expressions in a formal syntax. Inner nodes represent functions, whereas leaves represent terminal symbols. Together they form the alphabet of the program. Figure 2.3 shows an example of a syntax tree for the program $max(x + x, x + 3 * y)$, where the function set is $\{max, +, *\}$ and the terminal set is $\{3, x, y\}$.

An important property of the elements that compose the function set is its arity. The arity of a function is the number of arguments it receives, given that both functions and terminals can be considered. For example, the max function of Figure 2.3 has an arity of two.

The chosen alphabet must be complete, i.e., the set of functions and terminals must be wide enough to produce at least a solution for the problem. Sometimes this is difficult to meet because the solution is unknown. Another property that must be assured is the closure of the function set. This requirement has the goal to guarantee that any function of the set is able to receive as argument any other function or terminal, with no concerns about their type. Sometimes, typed functions or terminals exist, requiring special care not to break the closure property.

A problem arises from this kind of representation. Being individuals represented using a tree, and because trees of different individuals are different in size, with time this will lead to a huge growing in the candidates solution size, making the produced code not human-readable. This problem is known as bloat and can be prevented by defining a maximum tree size (forcing variation operators not to exceed it) or by penalising the fitness of individuals with relation to their size.

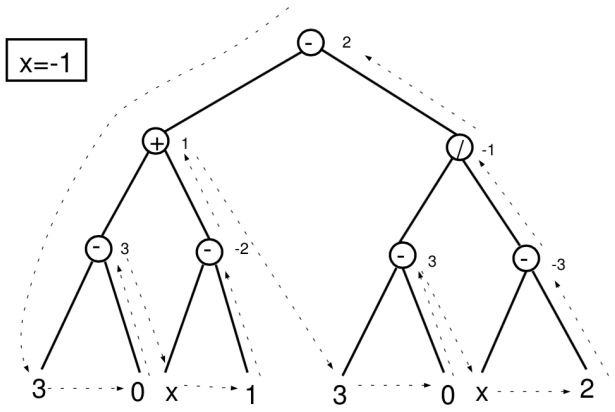


Figure 2.4: Tree interpretation procedure [65].

To decode trees, nodes are processed in a depth-first post-order way so that, when a node with arguments is analysed, the value of its arguments is already known. In other words, the tree is traversed recursively, starting from the root node, skipping for later processing all those nodes which children's value is still unknown. Figure 2.4 illustrates the decoding procedure.

Evaluation Function As in standard EAs, in GP, selection aims at measuring the adaptivity of the candidate solutions to the environment and it has to have the capacity of establishing a comparison between each pair of individuals.

Normally, when the optimum solution is known, the quality of an individual is measured by assessing his distance to it. In GP, because candidate solutions usually represent programs, this proximity to the optimum is calculated by executing it, giving, as input, sets of arguments and comparing the produced output with the correct solution for that set.

If no optimum is known, probably just a maximisation or minimisation with relation to some criteria may be being sought .

Like mentioned above, sometimes the complexity of the individual is incorporated in the calculation of his fitness (Equation 2.2). This is done to prevent the continuous increasing of the individual's size, also known as bloat. Preventing this growth will lead to programs easier to read. Complexity may be measured using, for example, the number of nodes in the tree.

$$fitness'(x_i) = fitness(x_i) - complexity(x_i) \tag{2.2}$$

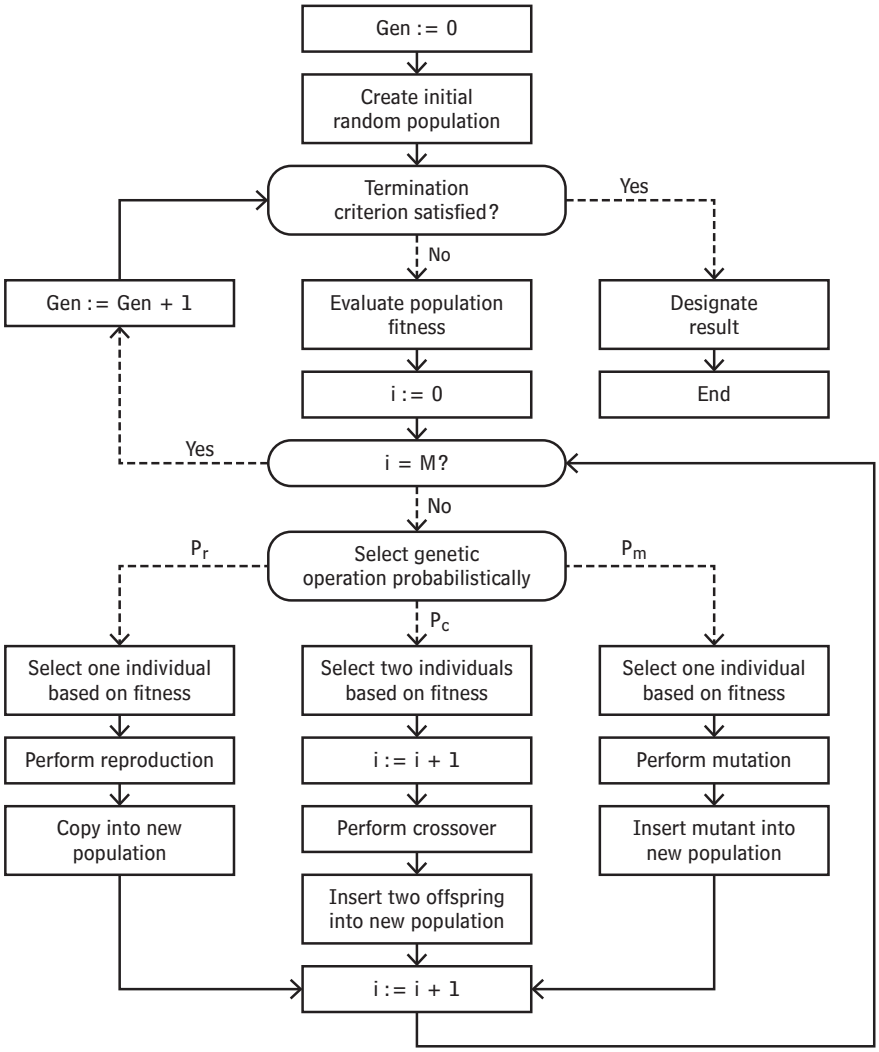


Figure 2.5: Typical GP Algorithm flow-chart, focusing the variation operators applied to each individual [32].

Variation Operators Like in any other form of EA, GP mutation and recombination aim at generating offspring. Applying mutation to a parent produces a single child and recombination takes two parents to form children. Although, the way they are implemented, at a lower level of detail, is different.

Whereas in typical EA approaches it is normal to apply to a candidate solution both mutation and recombination in the same iteration, in GP that does not usually happen. Operators are generally exclusive, being a normal rate for crossover around 90% and of 1% for mutation. For that, when the sum of crossover and mutation rates does not reach 100% another operator called reproduction is used. Reproduction consists only on the copy of a selected candidate solution to the offspring, without

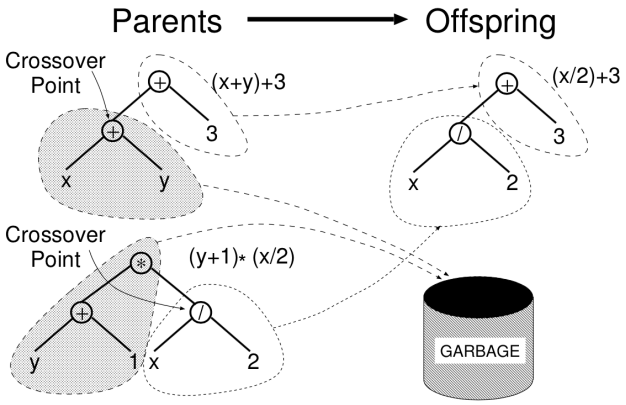


Figure 2.6: Example of subtree crossover [20].

altering its structure. A flow-chart depicting the choices made at the level of the variation operators is presented in Figure 2.5, where P_m represents the probability of mutation, P_c the probability of crossover and P_r the probability of reproduction.

Recombination When it comes to recombination there are at least two distinct approaches, diverging in the number of children produced by two parents previously chosen by a selection mechanism.

In one hand, [32] defends that the recombination of the genetic material from two parents should origin only one child. From each parent (A and B) a crossover point (a node of the individual's tree) is chosen (n_a and n_b). Then, the offspring is created by replacing the subtree starting at n_a by the subtree with root in n_b (see Figure 2.6). In the other hand, as mentioned in [18], the reverse process should be also done, that is, the subtree starting at n_b is also replaced by the one rooted in n_a, giving origin to two children. To preserve the parents, copies of them must be done and the operator applied to their copies.

A problem was noticed by Koza [32] regarding the choice of the crossover points. If given the same probability to all nodes, it would be much more likely to choose a leaf than an internal node, which would lead to the exchange of little genetic material. For that reason, as a rule of thumb, the probability of choosing internal nodes is typically set to 90%, leaving only 10% to leaves.

Mutation The most common way of mutating a tree is to replace one of its subtrees, starting at a randomly chosen node, by a randomly generated tree (see Figure 2.7). That can be accomplished as if a recombination was being done. Basically

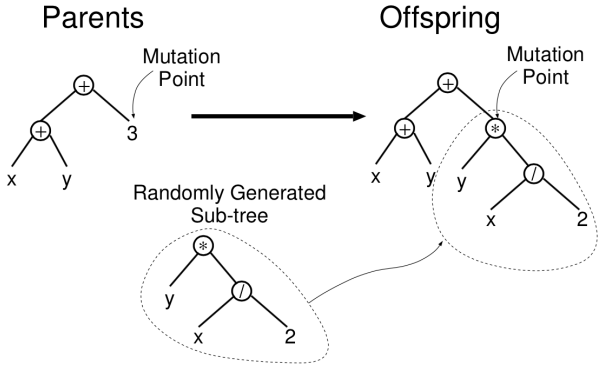


Figure 2.7: Example of subtree mutation [20].

the idea here would be to recombine the individual's tree with a randomly created one (generated in the same way as the initial population).

Other not so common mutations are: (i) node mutation, which consists in replacing a function node by another of the same arity; (ii) shrinkage mutation, aiming at the reduction of the tree, randomly chooses a node and then replaces it by one of its children, excluding all the others and their branches; (iii) permutation mutation, which changes the order of the arguments of a function.

Initialisation The creation of the initial population should preferentially cover all the domain, not favouring particular configurations, unless a-priori knowledge about the problem exists. Trees should be created in a stochastic way using the elements of the function and terminal sets, combining instances of both, recursively, giving origin to different tree lengths and complexities.

Before introducing the various methods that have been proposed to generate candidate solutions, the definition of depth must be presented. In a tree representation, the depth of a node is given by the number of edges that need to be traversed from the root node until the node of which the depth is sought. The depth of a tree corresponds to the depth of the deepest node.

Full Creation Method – The depth of all leafs is the same and equal to a specified value. That is, until the maximum depth is reached, more functions (internal nodes) are added, given that different functions require a different number of arguments. Then, terminal symbols are added (leaves).

Grow Creation Method – Unlike the previous method, it does not require the tree to be complete. If in a branch, a terminal symbol appears, that path is closed, not reaching the maximum depth.

Ramping Method – Consists in dividing the initial population in different sets, applying to each one the full creation method with a maximum depth that is in a range from the minimum real depth to the maximum real depth. For example, if it is pretended to create 30 candidate solutions with a minimum real depth of 2 and a maximum real depth of 4, there will be instanced, using the full creation method, 10 individuals with maximum depth 2, 10 individuals with maximum depth 3 and 10 individuals with maximum depth 4.

Ramped Half-and-Half Method – Based on the ramping method, divides the initial population in sets, splitting then each set into two. To the first half the full method is applied, the remaining candidate solutions are built with the grow method. For example, if we want to create 30 candidate solutions with a minimum real depth of 2 and a maximum real depth of 4, there will be instanced: 5 individuals with maximum depth 2 (full method), 5 individuals with maximum depth 2 (grow method), 5 individuals with maximum depth 3 (full method), 5 individuals with maximum depth 3 (grow method), 5 individuals with maximum depth 4 (full method) and 5 individuals with maximum depth 4 (grow method).

From all the above presented techniques, the most used for the creation of the initial population is ramped half-and-half, for being capable of creating diverse candidate solutions, with different tree sizes and shapes.

2.2.2.2 Parallel Distributed Genetic Programming

Proposed by Poli [62,63], Parallel Distributed Genetic Programming (PDGP) aims at a graph representation of programs. Each graph can have different sizes and shapes, within predefined limits.

A graph representation turns it possible to express a much bigger class of programs and to reuse nodes, consequently increasing efficiency.

As in Section 2.2.2.1, the components of the algorithm that are going to be approached are only the ones that differ from the standard EAs, presented in Section 2.2.1.

Representation The idea behind PDGP is that genotypes are represented by graphs, where the nodes stand for functions and terminals and where the links between them represent the flow of control and results.

As in Tree-Based GP (Section 2.2.2.1), the set of functions and terminals form the alphabet. The same properties have to be met: completeness and closure.

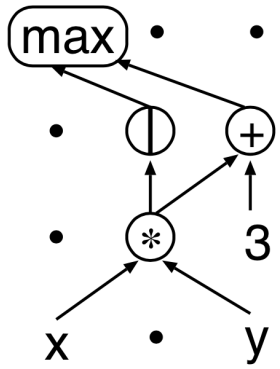


Figure 2.8: Graph representation for the expression $max(x*y, 3+x*y)$ [63] (adapted). To compare with a Tree-Based GP representation check Figure 2.3, that represents the same expression, as a tree.

Figure 2.8 depicts the typical PDGP representation. To each node a position in a pre-defined multi-dimension grid is assigned; additionally, connections between nodes are limited to be upwards. Another restriction is that these connections can only happen between nodes in adjacent lines. Function I is known as identity function and it works as a pass-through node, returning as output what it receives as input.

It is possible to add introns to the previously presented representation. Introns are unexpressed parts of the code; that is, despite being present, they are not used. This can be accomplished by placing, at every free position in the grid, a function or terminal, even if they are not directly or indirectly connected to the output.

Variation Operators Variation operators, as always in EAs, have the goal of producing offspring, based on the existing candidate solutions. In PDGP, recombination randomly picks two individuals as parents and produces one child. Mutation, from a single individual, produces a new one.

Recombination A simple form of recombination is based on the idea that sub-graphs are functional units whose output is used by other functions; for that, its flow must not be broken. Sub-graph Active-Active Node (SAAN) is described in [63] as the basic form of crossover. According to [63] the three main SAAN steps are:

1. A random active node is selected in each parent (crossover point);
2. A sub-graph including all the active nodes which are used to compute the output value of the crossover point in the first parent is extracted;

3. The sub-graph is inserted in the second parent to generate the offspring (if the x coordinates of the insertion node in the second parent is not compatible with the width of the sub-graph, the sub-graph is wrapped around).

Other methods of accomplishing crossover exist and are described in [63].

Mutation Mutation is very similar to the previously presented in Tree-Based GP. It is enough to randomly select a node that is not an intron, replacing it by a randomly generated sub-graph (global mutation).

Mutations affecting only the functions represented by each node do exist too (node-mutation). A specific mutation was also proposed, link-mutation, that aims at changing a randomly chosen graph connection.

Initialisation The first step in the creation of the initial population is to decide if balanced or unbalanced candidate solutions are pretended and if there are to be used introns. If balanced graphs are sought there has to be defined a maximum distance between the terminals and the output nodes, else they can occur at any random place. To complement, if introns are to be used, the grid can be filled with functions and terminals at random positions. When a function is added, a random number of links from that node to others must be also created.

Another possibility is to create individuals like in Tree-Based GP; that is, with root in the output nodes, recursively add functions and terminals, taking into attention the arity of each function. In this case, the answer to the initial question of whether or not balanced or unbalanced graphs are pretended is also needed, as the definition of the maximum diameter of the graph.

2.2.2.3 Cartesian Genetic Programming

Cartesian Genetic Programming (CGP) [49] is a graph-based form of GP introduced by Miller. In its basic form it aims at solving any problem where the solution can be represented by a direct acyclic graph. Further work has been done to extend this principle. In [83] the usage of modules (relevant blocks) is referred, [24] introduces the capacity of handling multiple data types to CGP, and in [25] self-modifying properties are featured.

In the following subsections the general form of CGP will be detailed, focusing the differences from other types of GP.

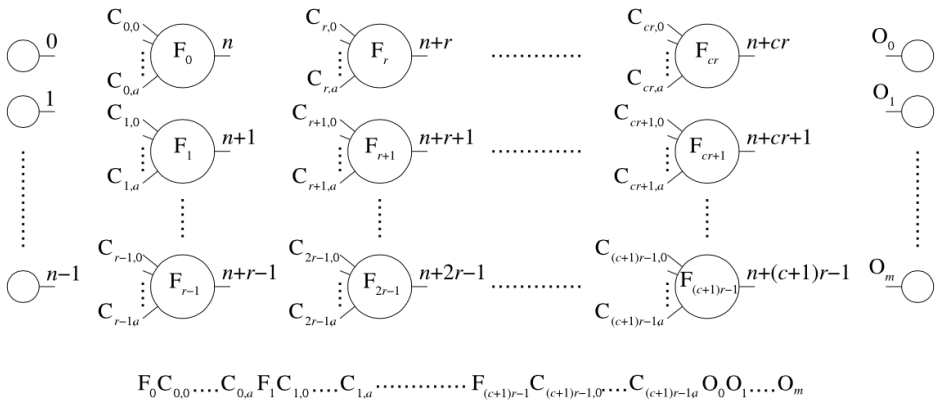


Figure 2.9: CGP generic representation [49].

Representation Standard CGP has its representation based on direct and feed-forward acyclic graphs. The nodes are then mapped into a set of integers, where the first one represents the identifier of the function and the others its inputs. For that, the number of integers needed to encode each node depends on the arity of the function, totalling $1 + arity_{function}$.

In the genotype of each individual there are considered two types of genes: function and connection genes. The first ones correspond to addresses in the function look-up table (table that establishes the correspondence between a function's label and a physical address), whereas the second one mentions where the node gets its inputs from (address in a data structure, normally an array, that stores the outputs of each node plus the program input addresses). So, in a more accurate manner, it is said that the previously referred integers correspond, in fact, to function and connection genes, respectively.

Prior to that, it is necessary to define the number of columns and rows of the grid of nodes and the number of levels-back, that constrains which columns a node can get its inputs from, i.e., the nodes prior to the current node from where it can get connections as input.

All the explained above is depicted in Figure 2.9, where all the flow from n_i inputs to n_o outputs is graphically detailed. The grid has n_r rows and n_c columns. Each node is identified by a function gene, F, and several connection genes, C, depending on the arity of each function. Data inputs and outputs are labelled consecutively as a way of guaranteeing its uniqueness. In the last line of the image the actual used genotype is shown.

For the fact that restrictions on output nodes are less rigid than in the remaining ones, it is possible that many of the genes are non-encoding, i.e., they are not ex-

pressed in the phenotype of the individual. In [50] their existence has been proven to be important.

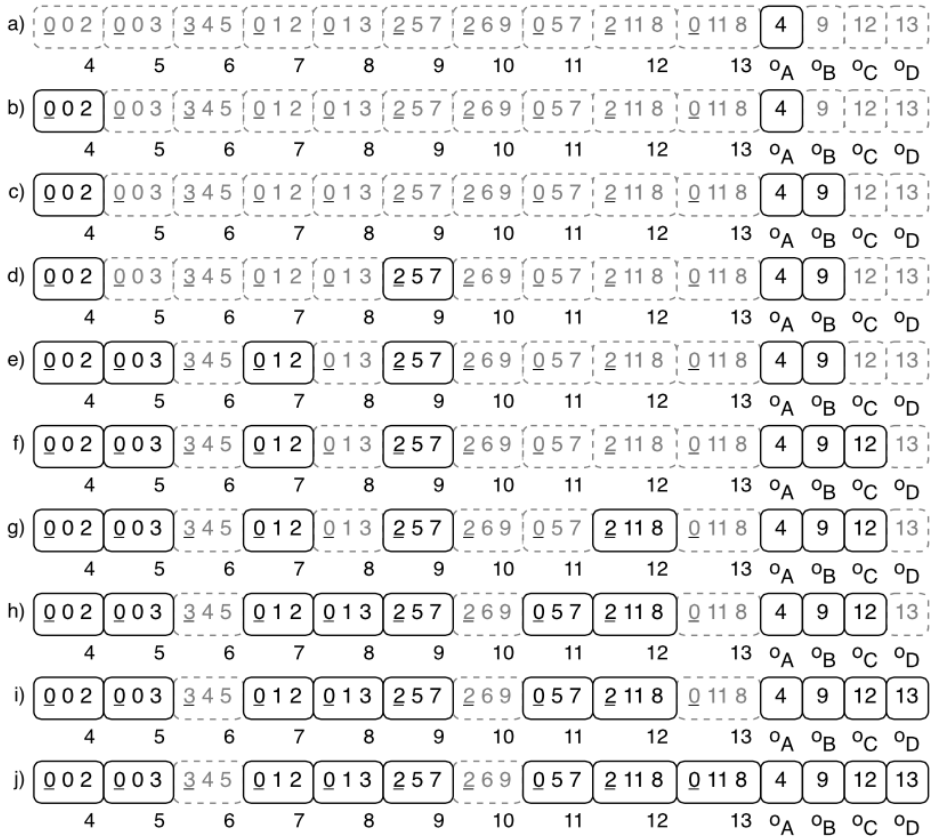


Figure 2.10: CGP decoding example [49].

Figure 2.10 represents the decoding algorithm. Starting from the output, it checks which nodes are directly addressed. Next, those nodes are analysed, to find out the nodes they are using. This process is repeated, recursively, until no more active nodes are found. It has the advantage of ignoring introns, not spending processing time with them.

Variation Operators Like in any other EA, CGP variation operators aim at generating offspring. Although, as will be discussed in the following subsections, the initial proposal of CGP only uses mutation. Some new crossover operators suitable to use with CGP have been later proposed, such as in [9].

Recombination Normally, when talking about standard CGP, recombination is not used. One point crossover was tested, although found to be disruptive. More recently, [9] proposed a new approach to recombination in CGP that has been proven

to speed up convergence. In this method, before applying crossover, the genotype of the individual has to be mapped into a float representation, where all values of genes lay in the interval $[0, 1]$ (check Section 4 of [9] for more details). Then, offspring is generated by applying Equation 2.3, where p_1 and p_2 represent the parents and r_i is a random generated number between 0 and 1.

$$o_i = (1 - r_i) * p_1 + r_i * p_2 \tag{2.3}$$

Mutation The mutation in basic CGP aims at changing a gene by any other random valid value. The validity concept appears due to the fact that genes are typed. If talking about a function gene it can only take values from the function's set addresses. Else, if talking about input genes, values from the output of any previous node or program inputs are possible.

This operator can be applied more than once in the same generation, to the same individual. The maximum number of times it can be applied (mutation rate), during the same generation is a-priori defined, as a percentage of the total number genes of the individual's genotype. Typically it is set to 1%.

2.3 Grammars

Before proceeding to grammars, an introduction to formal languages is presented [28, 35]. Formal languages are a way of representing a set of sentences that share the same core concepts. Their definition is formalised using a set of symbols and rules of formation that indicate how those symbols can be combined. Languages, at a higher level, are any subset of Σ^*, which represents the set of all the sentences that can be formed by combining zero or more symbols from the alphabet, Σ. It always contains the empty string, λ.

In the other hand, grammars aim at assessing if a particular sentence is or not correct. Formally, they are defined by a four-tuple (V, T, S, P), where:

- V is the set of variables, also known as non-terminal symbols;
- T is the set of terminal symbols;
- S is the start symbol and it has to be part of V;
- P is the set of production rules. They represent mappings of the type $x \to y$, $x \in (V \cup T)^+$ and $y \in (V \cup T)^*$.

The process of replacing a non-terminal by a given production is called derivation step and they are what lead to the expansion of grammars, from the start symbol. Derivation steps are applied until no non-terminal symbol is left for expansion.

For the language $L(G) = \{a^n b^{n+1} : n \geq 0\}$ [35], where $\Sigma = \{a, b\}$, an example of a grammar is $G = (\{S, A\}, \{a, b\}, S, P)$, with productions:

$S \rightarrow Ab$

$A \rightarrow aAb$

$A \rightarrow \lambda$

With the grammar of the previous example, the derivation of the sentence "aabbb" can be obtained by applying the following derivation steps:

$S \overset{S \rightarrow Ab}{\Longrightarrow} Ab \overset{A \rightarrow aAb}{\Longrightarrow} aAbb \overset{A \rightarrow aAb}{\Longrightarrow} aaAbbb \overset{A \rightarrow \lambda}{\Longrightarrow} aabbb$

Several types of grammars exist. In the following paragraphs focus will be given to Regular Grammars and Context-Free Grammars (CFGs). The main interest resides in CFGs, although those can be considered an extension of the first ones. A graphical type of grammars will also be presented: Shape Grammars.

Regular Grammars can be considered of two types: right or left-linear. If a grammar, $G = (V, T, S, P)$, has all production rules in the form $A \rightarrow xB$ or $A \rightarrow x$, $A, B \in V$ and $x \in T^*$, it is said to be right-linear. Otherwise, if the production rules are of the form $A \rightarrow Bx$ or $A \rightarrow x$ the grammar is said to be left-linear.

As seen, in the previous mentioned forms, the production rules can only have one non-terminal in its right side. For example the grammar with the following production rules: $A \rightarrow xB$; $A \rightarrow xB$ and $A \rightarrow xBB$ is not considered a regular grammar because: (i) it has production rules both in the right and left-linear forms; (ii) it has one production rule with more than one non-terminal in its right-side. Despite that, if violation (ii) was not present, the grammar could still be considered linear, meaning that at most one non-terminal symbol can occur on the right side of any production rule, without any restriction regarding its position.

From all the above, and specially from the example, it is easily concluded that a regular grammar is always linear but the opposite is not necessarily true, i.e., linear grammars are not required to be regular.

Context-Free Grammars (CFGs) are in all similar to regular grammars, removing all restrictions in the right side of production rules, allowing in this way the creation of more powerful grammars. For example, using a regular grammar it is not possible to model the parenthesis order of programming languages; a CFG can

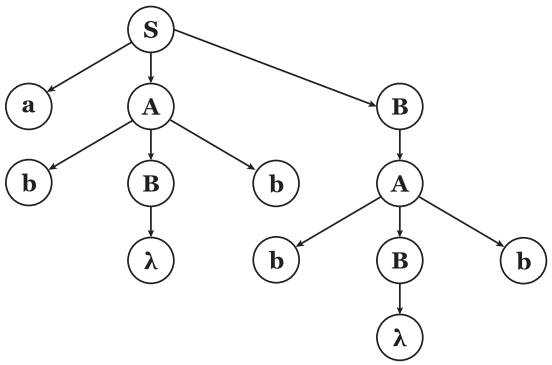


Figure 2.11: Example of a derivation tree for the sentence "abbbb" using grammar $G = (\{S, A, B\}, \{a, b, \lambda\}, S, P)$, with productions: $S \rightarrow aAB$; $A \rightarrow bBb$; $B \rightarrow A|\lambda$ [35].

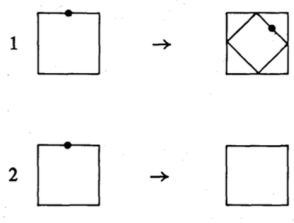
achieve it. Being that said, a CFG, $G = (V, T, S, P)$, has its production rules of the form $A \rightarrow x$, where $A \in V$ and $x \in (V \cup T)^*$. All regular grammars are CFGs too; the opposite is not true.

As production rules can have several non-terminals, a derivation is said to be leftmost if it is the leftmost non-terminal symbol that is replaced in each derivation step. An analogous procedure is taken for rightmost derivations, expanding, in each derivation step, the rightmost non-terminal symbol first.

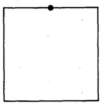
A way of representing derivations is by the use of derivation trees, where the nodes are labelled with terminal (inner nodes) and non-terminal symbols (leaf nodes). An example of the derivation tree for the sentence "abbbb" using grammar $G = (\{S, A, B\}, \{a, b, \lambda\}, S, P)$, with productions: $S \rightarrow aAB$; $A \rightarrow bBb$; $B \rightarrow A|\lambda$ is shown in Figure 2.11.

Given the following grammar $G = (\{E, I\}, \{a, b, c, +, *, (,)\})$ with productions: $E \rightarrow I$; $E \rightarrow E + E$; $E \rightarrow E * E$; $E \rightarrow (E)$; $I \rightarrow a|b|c$ it is possible to derive the expression "$a + b * c$" by at least two different ways [35]. This problem is known as ambiguity and it can be solved by rewriting the grammar. This is sometimes not enough because the ambiguity can be in the language itself. Other solution passes by establishing precedence rules. More about this forms of solving ambiguity can be read in [35].

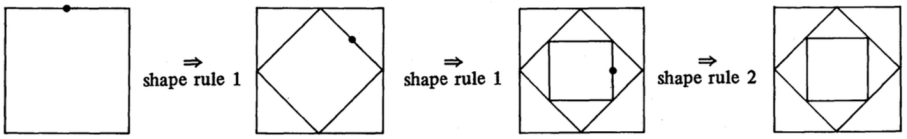
One of the main applications of CFGs is in the definition of programming languages and in the construction of interpreters and compilers. Some adaptations to it were performed, in order to improve readability, forming the Backus-Naur Form (BNF). In BNFs non-terminals are enclosed in triangular brackets, terminals do not use any special notation and ::= is used instead of $\rightarrow$. Those are the main differences



(a) Shape rules.



(b) Initial shape.



(c) Generation of a shape using the shape rules in (a) and the the initial shape in (b).

Figure 2.12: Shape grammars example [77].

between CFGs and BNFs, being the last one used as the standard for representing programming languages.

Procedures for simplifying CFGs do exist: (i) removal of λ production rules; (ii) removal of unitary production rules; (iii) removal of useless production rules and there are two main forms of normalisation: (i) Chomsky Normal Form; (ii) Greibach Normal Form. As these are not the focus of the present Dissertation, if pretended, further reading material can be found in [28, 35].

Shape Grammars were introduced by Stiny and Gips in a way similar to CFGs, but applied using shapes. Shapes are mentioned to as limited sets of lines that can be defined in a Cartesian coordinate system with real axes and an associated Euclidean metric, providing that no two distinct lines can be combined to form a single one [77].

Shape grammars consist on a four-tuple (S, L, R, I), where:

- S is the set of shapes (equivalent to T of CFGs);
- L is the of symbols (equivalent to V of CFGs);
- R describes the set of shape rules, in the form $a \rightarrow b$, where $a \in \{S, L\}^+$ and $b \in \{S, L\}^*$ (equivalent to P of CFGs);
- $I \in \{S, L\}^+$ refers to the initial shape (equivalent to S of CFGs).

An example of the application of a shape grammar is shown in Figure 2.12, where the shape rules are represented in 2.12a, the initial shape in 2.12b and a generation

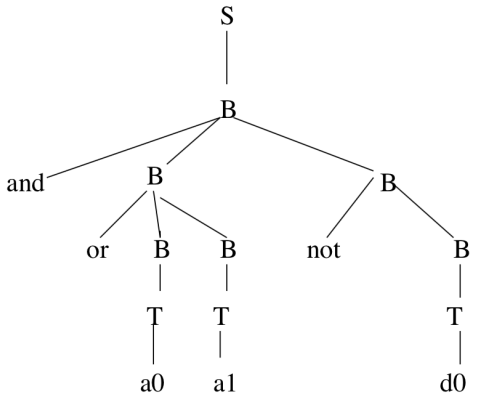


Figure 2.13: Example of a derivation tree [86].

of a shape using the grammar in 2.12c. It is easily seen that the used grammar is not deterministic as shape rule 1 can be applied an infinite number of times.

In the following subsections, methods that use CFGs with GP are going to be presented. In Section 2.3.1 the grammar itself will be evolved while, in Section 2.3.2, the GP evolution has its representation based on grammar mappings.

2.3.1 Grammar Evolution

In [84–86] Whigham proposed a method for evolving Context Free Grammars (CFGs), CFG-GP, based on Koza's tree GP approach.

One of the main disadvantages of the method introduced by Koza is the need to assure the closure property, which will make it possible for crossover and mutation operators to take place in any position of the tree, leading always to valid candidate solutions. CFGs can overcome the need to meet this property.

2.3.1.1 Representation

As mentioned previously, Whigham aimed at evolving CFGs and, as so, the most intuitive representation are derivation trees.

In Figure 2.13, a representation of the grammar formed by the following production rules is depicted:

S $\rightarrow$ B

B $\rightarrow$ and B B | or B B | not B | if B B B | T

T $\rightarrow$ a0 | a1 | d0 | d1 | d2 | d3

The non-terminals set is $\{and(2), or(2), not(1), if(3)\}$ and the terminals are $\{a0, a1, d0,$

$d1, d2, d3\}$. The values in brackets, used in the non-terminals, represent their arity.

2.3.1.2 Initialisation

CFG-GP creates its initial population with a method based of the Ramped-Half-and-Half from traditional Koza's GP. After defining the maximum depth, the following algorithm, from [86], is applied:

1. Label each production rule $A \rightarrow x$, where $A \in N$ and $x \in \{N \cup \Sigma\}^*$, with the minimum number of derivation steps to create only terminals;
2. For the range of depths and number for each depth $D = i...j$ do:
   (a) Select the start symbol, S, and label it as the current non-terminal A;
   (b) Randomly select a production $P_1 \in P$ of the form $A \rightarrow x$ with minimum derivation steps to $\Sigma^* < D$;
   (c) For each non-terminal $B \in x$, label B as the current non-terminal and repeat steps (b) and (c).

2.3.1.3 Variation Operators

Variation operators, as in any EA, aim at producing offspring. In CFG-GP both recombination and mutation are applied, with different probabilities. Recombination aims at, from two parents, producing offspring, whereas mutation is applied to a single candidate solution.

Recombination is constrained to happen in non-terminals and, in order to guarantee that offspring is valid, the recombination algorithm picks the same node in both parents as the crossover point. From [86] it is possible to get the algorithm, as follows:

1. Select two programs, with derivation trees p_1 and p_2, from the population, based on fitness;
2. Randomly select a non-terminal $A \in p_1$;
3. If no non-terminal matches in p_2, go to step 1;
4. Randomly select $A \in p_2$;
5. Swap the subtrees below these non-terminals.

6. If the maximum depth is exceeded, the whole process is aborted and it should be restarted at step 1.

Mutation in this approach is the same as in traditional tree-based GP. To start, a randomly non-terminal is chosen as the mutation point and then, a new random tree is generated (as in the initialisation procedure), having as start node the mutation point. Next, the new built tree is inserted in the mutation point, replacing all the previous structure. In any case, the maximum tree depth should not be exceeded.

2.3.1.4 Bias

Usually, CFGs lead to large search spaces. It was proven in [84] that, for a specific problem, the introduction of biases produce an improvement in the success rate of the algorithm. Different types of bias exist, being the three major: (i) selection bias; (ii) language bias; (iii) search bias. A deeper explanation regarding each one of them can be found in [85]. From that, Whigham inferred that the dynamic creation of new production rules, in run time, based on the genotypes of the fittest individuals probably would acquire better results. That was proven to be true.

To accomplish that, it is first needed to find the most suitable areas of individuals that conduct to new production rules and then incorporate these into the CFG.

To start, an individual is selected to extend the current grammar. The chosen individual is the one which has a higher fitness value. If two or more happen to have the same quality, the one with the least depth is picked. Then, the identification of the better suited productions, that may be incorporated into the CFG, is done, using a bottom-up method. From the leaf nodes (terminals) it climbs up the parse tree to the next production which has, in the same level, other terminals or non-terminals. The traversed path describes the new production rule, which will be included in the CFG. The terminal that is chosen to be propagated in the bottom-up approach is the deepest and left-most one. In Figure 2.14 it is possible to identify the new production rule as B $\rightarrow$ B if a0 B B (starting from a0).

Another concept that was proposed was the merit value, which essentially represents the probability of choosing a production rule while applying a derivation step. When starting the whole evolutionary process, and no bias exists in the CFGs, a default merit value of 1 is set to all production rules. Then, upon identifying suitable production rules (like depicted in Figure 2.14), if the identified production already exists in the production rules set, its merit is incremented by one. Otherwise, it is added to the set and its merit set to one.

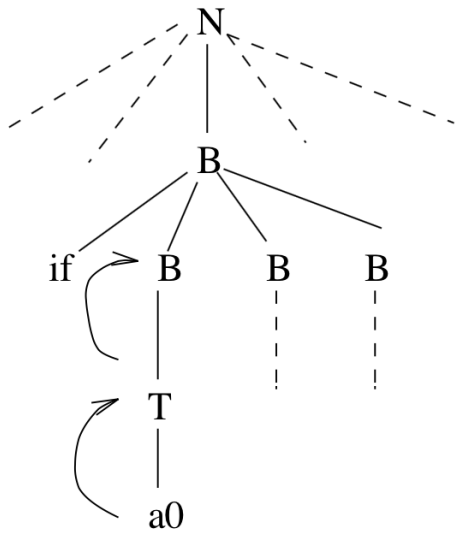


Figure 2.14: Identification of new production rules [86].

Changes were also proposed to the selection mechanisms, so that they would be proportional to productions merit instead of individuals fitness. Epoch-replacement[3] is used, producing, from generation to generation, individuals that incorporate this knowledge, which is acquired over time.

2.3.2 Evolution Based in Grammars

A method of promoting evolution based in grammars is proposed in [55, 56, 60] by O'Neill and Conor Ryan, which was named Grammatical Evolution (GE). It has been suggested as a way of evolving a program, in any language.

In the following subsections the main components of this evolutionary algorithm are going to be described. In particular, representation, variation operators and the decoding operation will be specified.

2.3.2.1 Representation

In GE, individuals are represented as variable-length binary strings of bits, where each group of 8 bits represents a codon, which denotes an integer. Those integers will them be mapped into BNF rules in order to allow program execution and consequent fitness assessment, as will be further detailed in Section 2.3.2.2.

From this type of linear representation, the generation of the initial population can be easily understood as the generation of randomly, length-variable strings of bits

[3]Between each generation, a percentage of the worst individuals is replaced by new randomly generated ones.

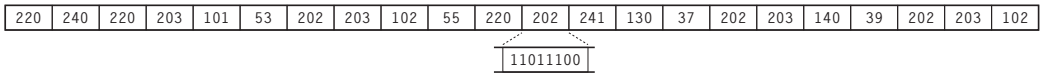


Figure 2.15: Individual's representation example [60] (adapted).

(ones and zeros), within a defined minimum and maximum number of codons (groups of 8 bits). An example of an individual's genotype is presented in Figure 2.15 where, for the sake of simplicity, instead of the binary representation its integer encoding is represented.

2.3.2.2 Decoding

Beginning by the start symbol, the decoding procedure reads each codon in order to decide which derivation step should be applied. Derivations follow a leftmost approach, i.e., the derivation that is picked to be expanded is the first non-terminal symbol. In order to decide with which derivation rule should the program be expanded the mapping function shown in Equation 2.4 is used, where *codon* is the integer value resulting from the 8 bits and *non_terminal_rules* is the number of production rules, in the BNF, that the node that is to be expanded has. This process is repeated, recursively, until no more non-terminals exist.

$$rule = codon\ MOD\ non\_terminal\_rules \tag{2.4}$$

An example of the application of this algorithm will be now presented, using as genotype the genetic material of Figure 2.15 and the grammar G =({*expr*, *op*, *preop*},{*Sin*, +, −, /, ∗, *X*, 1.0, (,)},<expr>,*P*), from [60], where the production rules set is:

(1) <expr> ::= <expr><op><expr> (0)
| (<expr><op><expr>) (1)
| <preop> (<expr>) (2)
| <var> (3)

(2) <op> ::= + (0)
| − (1)
| / (2)
| ∗ (3)

(3) <preop> ::= *Sin*

(4) <var> ::= X (0)
| 1.0 (1)

From the starting symbol, <expr>, that has 4 possible production rules, and the codon representing 220 are read. As 220 MOD 4 = 0, from <expr>, <expr><op> <expr> is originated. In the second step, with the leftmost non-terminal being <expr> the operation 240 MOD 4 will be performed. The result is 0 and for that <expr><op><expr><op><expr> is reached. This process is repeated until no further non-terminal is present in the sentence. If continued, the reader will notice that the final expression will be given by $1.0 - Sin(X) * Sin(X) - Sin(X) * Sin(X)$.

If, for any reason, during the decoding process no more codons are available, that is, if all of them are used, there can be performed one of the following actions. In one hand, the individual can be declared invalid, and later replaced by the steady-state replacement mechanism. In the other hand, wrapping can be done, i.e., genetic material is reused, re-starting reading the codons from the initial one. The opposite can also happen. It is possible that not all the genetic material is used. No concerns should be given to that situation. Individual's can also be declared invalid for being unable to produce a complete mapping, as their grammar can be recursive.

With regard to the reading of codons, to one condition an exception should be made. If expanding a non-terminal that has only a production rule, no codon should be read, as only one outcome is possible. Returning to the previous example, if a decision was needed to me made regarding the non-terminal <preop> it is obvious that it was going to be replaced by the terminal Sin as no other option exists, as so, no codon should be used.

The procedure herein represented is deterministic. If this process is repeated more than once, to the same individual, as the codons order is the same the produced output will always be equal.

To end this section, redundancy of the decoding/mapping method must be addressed. By mathematical reasons, the value of $(A * B)$ MOD B outputs always the same result for any integers A and B. Placing this in context with the mapping technique under study, it becomes clear that several paths do exist that lead to the same final expression. That should not be seen as a problem as it also happens in nature, with proteins having multiple possible nucleotides encodings. In [59] this redundancy has been proven to preserve population's diversity, improving in this way GE's efficiency.

2.3.2.3 Variation Operators

For the fact that GE uses a linear form of representation, the applied variation operators are the typical ones for this kind of approaches. If, during the application of this operators, an invalid individual is generated, its fitness should be penalised and due to a steady-state replacement they will be likely replaced, whereas highly fitted individuals will be kept.

Recombination Standard GE employs single point crossover, i.e., after choosing the crossover points, the genetic material is exchanged between the two parents, giving origin to offspring, in this case two new individuals. Those new generated individuals then replace their parents in the population's pool.

For example, if considering two individuals as parents, represented respectively by 0101|01 and 1110|00, where | represents the crossover point, the offspring will be 010100 and 111001.

Further crossover techniques that might be suited to use with GE were also studied. The obtained results are fully described in [57] and demonstrate that other approaches were found to be no better than the addressed in the present section.

Mutation Several different mutations can be applied to linear representations; the ones herein presented are from [56] and [60].

In [60], point mutation and codon duplication are referred. Point mutation aims at changing a bit value, in a given position. In the other hand, duplication aims at randomly choosing a number of codons to duplicate, placing them at the end of the individual's genotype. [56] adds pruning to the previously mentioned operators. This last mutation aims at removing all the genes that are not used to express the characteristics of a given individual, that is, all those that are not used in the genotype to phenotype mapping procedure (detailed in Section 2.3.2.2).

Due to the fact that many introns may exist, neutral mutations can arise, i.e., even upon the application of mutations to individuals, their phenotype and, consequently, their fitness may suffer no modifications. This is potentially useful as it provides the individual with the capacity to cope with some potentially destructive mutations.

2.4 Conclusions

In the present chapter we started by presenting Evolutionary Computation as a way of linking computational power with the principles that guide evolution in nature. Then,

we drilled down over Evolutionary Algorithms, describing what they are and their main components, focusing later on one of its branches known as Genetic Programming. Regarding GP, multiple approaches to it were investigated, namely, Koza's Tree-based GP, Parallel Distributed GP and Cartesian GP, the last two encode individuals as graphs. To end, grammar theory and GP approaches that make use of them were detailed and explored. Some of the considered techniques just use grammars to guide evolution while others evolve the grammar itself.

All this overview and knowledge about such a wide range of ideas and notions kept us thinking about some questions and issues that serve as starting point for the current Dissertation. Is it possible to come up with a tool capable of dealing with multiple problem environments and domains? Can grammar formulations help in this task? If so, how? Which way of representing the solutions is more easily adaptable and interpretable? Is it worth trying to combine multiple components of the different approaches previously described?

During the following chapters we will provide some answers to the above questions, ending with a broad discussion of them, in Chapter 7.

Chapter 3

Representation and Operators

In the present chapter, approaches aiming at solving some of the raised questions in Section 2.4 are going to be addressed.

During the first stage of the Dissertation, and in line with previously developed work, a graph-based evolutionary engine was developed (Section 3.1). The objective of this approach is to evolve graphs resembling grammar structures. In initial work the graphs represented Context Free Design Grammars [11] only; later, the representation was generalised to cope with any type of grammar.

Later on, a tree-based evolutionary engine is proposed (Section 3.2). With the objective of making the task of generalisation to other domains easier, individuals are represented as derivation trees of a pre-defined grammar. Because this grammar specifies the form that can be taken by the grammar that will ultimately define the individuals, we can call it a pre-grammar, using some of the concepts introduced by Nicolau in [53].

In each of the above referred sections (Sections 3.1 and 3.2) the used representation and operators (population initialisation, mutation and crossover) are thoroughly detailed. As fitness assessment is similar in both approaches, we decided to merge its description in Section 3.3. To end, conclusions regarding the theory behind both methodologies are drawn (Section 3.4).

3.1 Graph-Based

The first tried approach for the evolution of grammar formulations is herein described. Individuals are represented as graphs, where each node encodes a production rule. Later, they are mapped into phenotypes and their quality assessed.

The next sections are organised as follows. Section 3.1.1 describes the representation used to encode the individuals genotype; random population initialisation is

```
start non_term_A

rule non_term_A{
   term_B {param_a 2, param_b 0}
   non_term_b{}
}
rule non_term_B 0.5{
  non_term_B{}
  term_B{param_b -1}
}
rule non_term_B 0.7{
  non_term_B{param_a 1}
  term_A{}
}
```

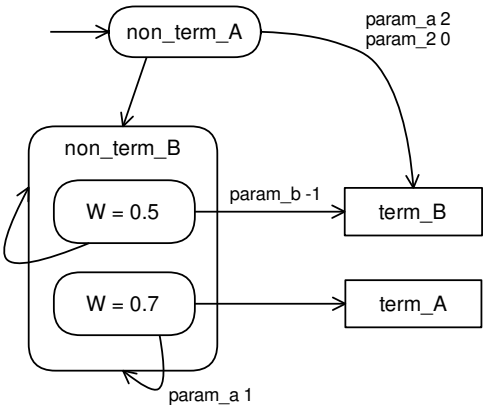


Figure 3.1: On the left a generic grammar; on the right, the same grammar represented as a graph. W represents the node's weight. Rectangles represent terminals and rounded rectangles non-terminals.

detailed in Section 3.1.2. Finally, Sections 3.1.3 and 3.1.4 respectively present the crossover and mutation operators.

3.1.1 Representation

In this approach each individual is represented by means of a directed graph formed by a set of nodes and links. Each node corresponds to a symbol (terminals and non-terminals) and each connection to a link between nodes.

Figure 3.1 depicts an example of a generic grammar and its corresponding graph representation. Individuals are interpreted starting from the starting symbol (in this case, non_term_A) and proceeding in a breadth-first fashion, expanding the production rules. Augmented grammars are allowed, i.e., it is possible to associate parameters with each symbol call, which are kept in the connection links between nodes (for example, the parameter $param\_a$ used in the call to $term\_B$ from non_term_A). The interpretation ends when there are no non-terminal symbols left to expand.

In order to allow the use of non-deterministic grammars, the definition of the same non-terminal rule more than once is allowed (non_term_B in Figure 3.1). Then, prior to its expansion one of them must be chosen, resorting to a stochastic process, where the weight of each node is used. Node weights can be found in each rule definition, right after the non-terminal name (see Figure 3.1); if no value is specified, a default of 1 is assumed. As one may notice, node weights can sum up more than 1; although, before choosing one node from the multiple possibilities, their probability is re-weighted so that their sum totals 1.

From the example it is also clear that loops may occur (*non_term_B* in Figure 3.1).

3.1.2 Random Initialisation

The creation of the initial population for the current evolutionary engine is of huge importance, being responsible for generating the first genetic material that will be evolved through time. Instead of having to supply to the system the first population, either composed of human-created grammars [41] or of a single minimal grammar [42], we decided for a procedure that allows the creation of a random population of individuals.

Algorithm 2 Random initialisation of a graph-based individual.

```
procedure RandomInitialisation(terminals, prob_t)
    min_v, max_v ← minimum, maximum number of non-terminal symbols
    min_p, max_p ← minimum, maximum number of production rules per non-terminal
    min_c, max_c ← minimum, maximum number of calls per production
    nonterminals ← RandomlyCreateNonTerminalSet(min_v, max_v)
    for all V ∈ nonterminals do
        numberofproductions ← random(min_p, max_p)
        for i ← 1, numberofproductions do
            productionrule ← NewProductionRule(V)
            numberofcalls ← random(min_c, max_c)
            for j ← 1, numberofcalls do
                if random(0, 1) < prob_t then
                    productionrule.InsertCallTo(RandomlySelect(terminals))
                else
                    productionrule.InsertCallTo(RandomlySelect(nonterminals))
                end if
                productionrule.RandomlyInsertProductionRuleParameters()
            end for
        end for
    end for
    individual.setProductionRules(productionrules)
    individual.RandomlySelectStartNode(nonterminals)
    return individual
end procedure
```

In simple terms, the method for creating a random candidate solution can be described as follows: we begin by randomly determining the number of non-terminal symbols and the number of production rules for each of the symbols (i.e., the number

of different options for its expansion). Since this defines the nodes of the graph, the next step is the random creation of connections among nodes and calls to terminal symbols. The parameters associated with the calls to terminal and non-terminal symbols are also established randomly. Finally, once all productions have been created, we randomly select a starting node, from the non-terminals set. Algorithm 2 details this process, which is repeated until the desired number of individuals is reached. Important to mention that the method $RandomlyCreatdeNonTerminalSet$ just creates random symbols, without any constraints or need to match them to anything else. More technically, it just creates a set of strings that become non-terminals.

3.1.3 Crossover Operator

Regarding the crossover operator, the rationale was to develop a method that would promote the meaningful exchange of genetic material between individuals. Given the nature of the representation, this implied the development of a graph-based crossover operator that is aware of the structure of the graphs being manipulated. The proposed operator can be seen as an extension of the one presented by Pereira et al. [61]. In simple words, it allows the exchange of subgraphs between individuals.

The crossover of genetic code between two individuals, a and b, implies: (i) selecting one subgraph from each parent; (ii) swapping the nodes and internal edges of the subgraphs, i.e., edges that connect two subgraph nodes; (iii) establishing a correspondence between nodes; (iv) restoring the outgoing and incoming edges, i.e., respectively, edges from nodes of the subgraph to non-subgraph nodes and edges from non-subgraph nodes to nodes of the subgraph.

Subgraph selection – Randomly selects for each parent, a and b, one crossover node, v_a and v_b, and a subgraph radius, r_a and r_b. Subgraph s_{ra} is composed of all the nodes, and edges among them, that can be reached in a maximum of r_a steps starting from node v_a. Subgraph s_{rb} is defined analogously. Two methods were tested for choosing v_a and v_b, one assuring that both v_a and v_b are in the connected part of the graph and one without restrictions. The radius r_a and r_b are randomly chosen, between 0 and the maximum diameter of the graph.

Swapping the subgraphs – Swapping s_{ra} and s_{rb} consists in replacing s_{ra} by s_{rb} (and vice-versa). After this operation the outgoing and the incoming edges are destroyed. Establishing a correspondence between nodes, repairs these connections.

Algorithm 3 Traversing the minimum spanning trees of two subgraphs.

```
procedure TRAVERSE(a, b)
    set_correspondence(a, b)
    mark(a)
    mark(b)
    repeat
        if unmarked(a.descendants) ≠ NULL then
            next_a ← RandomlySelect(unmarked(a.descendants))
        else if a.descendants ≠ NULL then
            next_a ← RandomlySelect(a.descendants)
        else
            next_a ← a
        end if
        **** do the same for next_b ****
        traverse(next_a, next_b)
    until unmarked(a.descendants) = unmarked(b.descendants) = NULL
end procedure
```

Correspondence of nodes – Let s_{ra+1} and s_{rb+1} be the subgraphs that would be obtained by considering a subgraph radius of $r_a + 1$ and $r_b + 1$ while performing the subgraph selection. Let mst_a and mst_b be the minimum spanning trees (MSTs) with root nodes v_a and v_b connecting all s_{ra+1} and s_{rb+1} nodes, respectively. For determining the MSTs all edges are considered to have unitary cost. When several MSTs exist, the first one found is the one considered. The correspondence between the nodes of s_{ra+1} and s_{rb+1} is established by traversing mst_a and mst_b, starting from their roots, as described in Algorithm 3.

Restoring outgoing and incoming edges – The edges from a $\notin s_{ra}$ to s_{ra} are replaced by edges from a $\notin s_{rb}$ to s_{rb} using the correspondence between the nodes established in the previous step (e.g. the incoming edges to v_a are redirected to v_b, and so on). Considering a radius of $r_a + 1$ and $r_b + 1$ instead of r_a and r_b in the previous step allows the restoration of the outgoing edges. By definition, all outgoing edges from s_a and s_b link to nodes that are at a minimum distance of $r_a + 1$ and $r_b + 1$, respectively. This allows us to redirect the edges from s_b to b $\notin s_b$ to a $\notin s_a$ using the correspondence list.

An example of the application of the above detailed crossover operator, from [42], is shown in Figure 3.2; on the left the parents and on the right the offspring. It is considered that $v_a = B$, $v_b = 1$ and $r_a = r_b = 2$. Subgraphs s_{ra} and s_{rb} are represented by the grey nodes and dotted edges stand for the MSTs. It is also assumed that

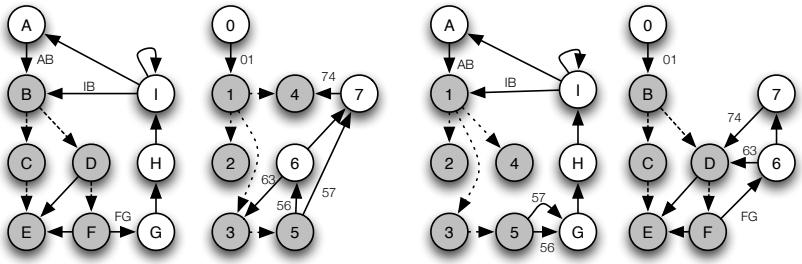


Figure 3.2: Graph-based crossover example [42].

Algorithm 3, because of its stochastic nature, returns the correspondence list {B-1, C-2, E-2, D-3, F-5, G-6, G-7, D-4}.

3.1.4 Mutation Operators

The mutation operators were designed to attend two basic goals: allowing the introduction of new genetic material in the population and ensuring that the search space is fully connected, i.e., that all of its points are reachable from any starting point through the successive application of mutation operators. This resulted in the use of a total of ten operators, which are succinctly described on the following paragraphs.

Startshape mutate – randomly selects a non-terminal as starting symbol.

Replace, Remove or Add symbol – when applied to a given production rule, these operators: replace one of the present symbols with a randomly selected one; remove a symbol and associated parameters from the production rule; add a randomly selected symbol in a valid random position. Note that these operators are applied to terminal and non-terminal symbols.

Duplicate, Remove or Copy & Rename rule – these operators: duplicate a production rule; remove a production rule, updating the remaining rules when necessary; copy a production rule, assigning a new randomly created name to the rule and thus introducing a new non-terminal.

Change, Remove or Add parameter – as the name indicates, these operators add, remove or change parameters and parameter values. The change of parameter values is accomplished using a Gaussian perturbation.

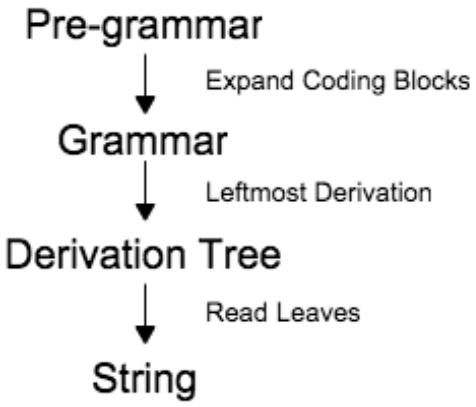


Figure 3.3: From pre-grammars to strings.

3.2 Tree-Based

With the objective of turning generalisation into an easier step, a methodology where the user can input the exact structure that the outputs of the system should follow was addressed. Keeping in mind that rationale, it is easily understood the choice for a tree approach, where each individual will represent a derivation tree of a pre-defined grammar. In this way, individuals are themselves derivations from a higher-level grammar.

Analogously to the previous section, after introducing what are pre-grammars and how they were used (Section 3.2.1) we will, in Section 3.2.2, detail the individual's representation. Then, we will move to the implementation details of the population initialisation, crossover and mutation operators, respectively in Sections 3.2.3, 3.2.4 and 3.2.5.

3.2.1 Pre-grammars

As mentioned earlier, in this type of approach the user is required to input a structure similar to a BNF, which specifies how the derivation tree should be expanded. It cannot be considered a BNF because some of the used symbol representations are not permitted in that form of representing grammars. In fact, this structure is still not the one that will ultimately be used for creating the derivation trees. It only specifies how must that grammar be formed, allowing for that, in addition to BNFs, the use of coding blocks, which are explained in the next paragraphs. As so, we call it pre-grammar, following notation similar to the one introduced by Nicolau and Dempsey, in [53].

Figure 3.3 depicts the steps that are needed to produce a derivation tree. As shown, first the pre-grammar (structure input by the user) must be converted into the actual grammar. Only then it is possible to generate random derivation trees. Later on, the leaves of the derivation tree are read (from left to right) and their values concatenated.

In the following paragraphs, to simplify writing, production rules of the pre-grammar will be referred to as pre-rules. Production rules of the actual grammar, i.e., the one derived from the pre-grammar, will be called grammar-rules.

The main difference between pre-grammars and the actual grammars resides on the existence of special coding blocks, that need to be further expanded. More precisely, we developed three coding blocks:

> ***<non-term>(x, y)*** – when mapping from the pre-rule to the grammar-rule, this block should be replaced by a random number of calls (between x and y) to the non-terminal <*non-term*>;
>
> ***<non-term$_1$>(x, non-term$_2$, type, bool)*** – implies that this block is to be replaced by x <*non-term*$_1$> non-terminals and that, in each one, the non-terminal <*non_term*$_2$> is to be assigned a random value of type *type*. Moreover, if *bool* is set to $True$ the random values generated should be kept, in order to create a new grammar-rule, <*non-term*$_2$>, with them;
>
> ***[x, y]*** – when mapping from a pre-rule into a grammar-rule nothing is changed in this block. It encodes a range, between x and y and later, while generating the derivation tree, if a grammar-rule with this encoding is found, it provides a value between the defined interval. Above all, it acts like a terminal symbol.

```
<non_term_A> ::= <non_term_B>(2, non_term_C, string, True)
                 <non_term_C>(0,3)
<non_term_B> ::= <non_term_C> [0,1]
```

Grammar 3.1: Example of a pre-grammar.

An example of an expanded grammar is shown in Grammar 3.2, which results from the pre-grammar depicted in Grammar 3.1. It is assumed that the two randomly created strings are "a" and "b" and that the random value 1 is returned (to perform the expansion of the second block). In the first grammar-rule of Grammar 3.2 the non-terminal *non_term_C*, called from the two *non_term_B* non-terminal rules, should derive the terminals a and b, respectively.

```
<non_term_A> ::= <non_term_B> <non_term_B> <non_term_C>
<non_term_B> ::= <non_term_C> [0,1]
<non_term_C> ::= a | b
```

Grammar 3.2: Grammar that results from the expansion of the pre-grammar presented in Grammar 3.1.

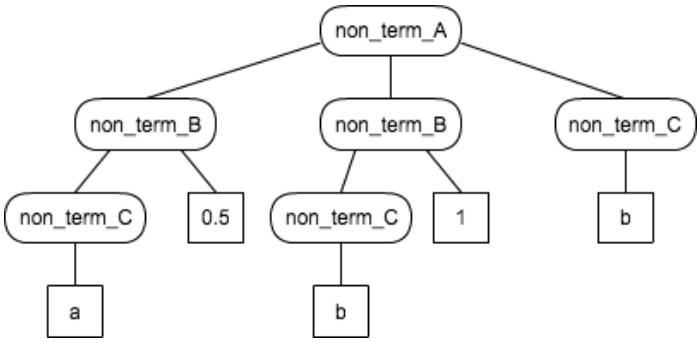


Figure 3.4: Example of a possible derivation tree of Grammar 3.2. Squares represent terminals and rounded rectangles non-terminals. To check the steps need to build the derivation tree refer to Figure 3.5.

3.2.2 Representation

As the name suggests, in this approach individuals are encoded as trees, more specifically, as derivation trees. Therefore, each inner node represents a non-terminal rule, i.e., a rule that still has to be expanded, whereas leaves stand for terminal symbols, which require no further processing.

Moreover, the proposed method aims at dealing with different domains. In order to make that possible, it receives as input a pre-grammar, as explained in Section 3.2.1.

An example of a derivation tree, generated using Grammar 3.2, is depicted in Figure 3.4. To get the production encoded in a derivation tree it is just needed to read all the leaves, discarding internal nodes. To do so, the derivation tree must be traversed in a depth-first pre-order way. In this method, after arriving to a new node, it expands its children nodes from left to right, until a terminal node (leaf) is reached. From that, it is possible to conclude that the derivation tree of Figure 3.4 gives origin to: "a 0.5 b 1 *b*".

3.2.3 Random Initialisation

In order to randomly create the initial population that will feed the evolutionary process it is first necessary to define the desired BNF grammar as well as the start

Algorithm 4 Random initialisation of a tree individual.

```
procedure TreeRandomInitialisation(TreeNode, BNF)
    if TreeNode is not terminal then
        expansion ← ChooseExpansion(TreeNode, BNF)
        for symbol ∈ expansion do
            newnode ← NewTreeNode(symbol)
            TreeRandomInitialisation(newnode, BNF)
            TreeNode.addChild(newnode)
        end for
    end if
end procedure
```

symbol. Then, beginning with the root symbol the derivation trees will be expanded, recursively, in a leftmost order, until no non-terminal symbol is left.

Algorithm 4 details the process. Taking into account that each symbol (terminal and non-terminal) is stored in a tree node, it expands the non-terminals from left to right, until a non-terminal is reached. The expansion rule is chosen randomly, from the set of possibilities, a-priori defined by the user. The algorithm must be repeated until the desired number of individuals is reached.

An example of the application of the above mentioned algorithm with the grammar introduced in Grammar 3.2 is shown in Figure 3.5.

3.2.4 Crossover Operator

As a result of the tree nature of representation, this operator can be easily understood as the typical one for this kind of approach, as previously detailed in Section 2.2.2.1, with the addition of some restrictions.

In order to produce offspring (off_a, off_b) from two parents (par_a, par_b) it is first needed to choose the cutting points in each tree (cut_a,cut_b) and then swap them. Because we are dealing with derivation trees, it is necessary to assure that the cutting points correspond to the same production rule; if not, the produced individuals would likely be ungrammatical. Moreover, the probability of choosing the cutting points from leaf nodes ($prob_{leaves}$) should be also specified; otherwise, as leaves tend to be more numerous than inner nodes, the change of genetic material would be, most of the times, minimal. Algorithm 5 details the above procedure.

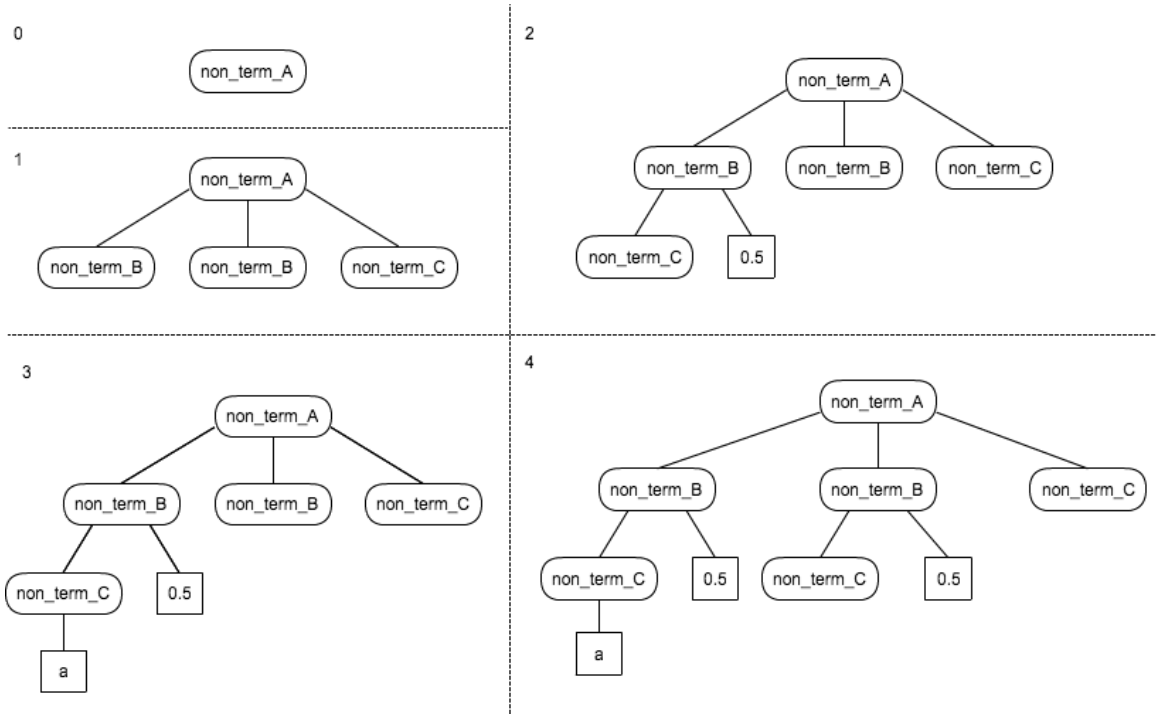


Figure 3.5: Random derivation tree creation, using Algorithm 4. For simplification, only the first 5 steps of the algorithm are depicted. Squares represent terminals and rounded rectangles non-terminals.

Algorithm 5 Crossover in tree-based individuals.

procedure TREECROSSOVER(par_a, par_b, $prob_{leaves}$)
  $\text{off}_a \leftarrow Copy(par_a)$
  $\text{off}_b \leftarrow Copy(par_b)$
  if $random() \leq prob_{leaves}$ **then**
    $nodes_a \leftarrow FindLeafNodes(off_a)$
    $nodes_b \leftarrow FindLeafNodes(off_b)$
  else
    $nodes_a \leftarrow FindInnerNodes(off_a)$
    $nodes_b \leftarrow FindInnerNodes(off_b)$
  end if
  $(cut_a, cut_b) \leftarrow ChooseCuttingPoint(nodes_a, nodes_b)$
  $\text{off}_a[cut_a] \leftarrow cut_b$
  $\text{off}_b[cut_b] \leftarrow cut_a$
  return $(\textbf{off}_a, \textbf{off}_b)$
end procedure

3.2.5 Mutation Operator

Mutation over derivation trees is a simple technique, resembling aspects from both crossover and population initialisation operators. First, a subtree of the individual has to be picked; once again probabilities for avoiding the selection of terminal symbols only are applied. Next, a new subtree to replace the previously chosen one is generated, as in Algorithm 4, taking as starting point the root of the subtree being mutated.

3.3 Fitness Assignment

In order to allow the use of the evolutionary algorithm in multiple domains, fitness assignment must be kept as a separate module. In simple words, the fitness function provided by the system's user receives as input an individual that represents a grammar (graph/tree), performs its mapping to a phenotype and then returns its quality value.

Examples of the assessment of fitness over two different domains are addressed in Chapter 4.

3.4 Conclusions

With the current chapter we aim at presenting the technical specifications of the developed GP approaches, which are going to be used from now on.

We started by introducing a graph-based evolutionary engine, where individuals are represented by nodes connected through a set of edges. Nodes mean production rules, whereas the links between them stand for the control of flow and parameters. On the other hand, we have an approach where the candidate solutions are derivation trees, associated with a pre-defined grammar. In this situation, inner nodes correspond to non-terminals of the pre-grammar and leaves to terminals.

Regarding the used operators, it is important to notice that they obviously are different because they focus on manipulating different types of structures. However, the tree-based operators are much easier to implement and understand than the graph-based ones. For instance, focusing on mutation, the graph-based mutation is comprised of ten operators, while the tree-based only has one which, in turn, is very similar to the initialisation method.

The impact of the different approaches as well as a detailed analysis of their operators will be carried out during the next chapter.

Chapter 4

Experimentation

After introducing, in the previous chapter, the theoretical concepts behind the proposed approaches, it is our intention now to design experiments that confirm their validity and capacity to evolve what is desired.

To prove that the methods are capable of dealing with different domains, i.e., that they have sufficient generalisation capacities we will apply them to the creation of images and musical sequences, where both of them result from the mapping from a grammar genotype, represented as a graph or derivation tree, to a phenotype (image or music).

In Section 4.1 the performance of tests for the evolution of Context Free Design Grammars [11] will be carried out. Both graph and tree-based approaches will be tested, independently. Later, a standardisation that merges components from both forms of representation will be investigated and tried. From the results of that last step we will then proceed to conducting experiments for the evolution of MIDI [1] files, leading this way to the output of musical sequences (Section 4.2). To end, in Section 4.3 conclusions will be drawn.

4.1 Evolving Context Free Art

Context Free Design Grammar (CFDG) [11] is a language capable of representing images through a compact set of production rules. In essence, a CFDG is an augmented context free grammar represented by a 4-tuple (V, Σ, R, S), where:

- V is a set of non-terminal symbols;
- Σ is a set of terminal symbols;
- R is a set of production rules that map from V to $(V \cup \Sigma)^*$;

```
startshape Edera

rule Edera {
     CIRCLE    {s 5}
     Ciglio    {}
     Edera     {x -5 y -1 s 0.90} }

rule Ciglio {
     SQUARE    {hue 200 sat 0.5}
     Pelo      {r 5 hue 200 sat 0.5}
     Ciglio    {y -1 r 0.5 s 0.998 b 0.005} }

rule Ciglio {
     SQUARE    {hue 200 sat 0.5}
     Pelo      {r 5 hue 200 sat 0.5}
     Ciglio    {y -1 r 0.5 s 0.998 b 0.005 flip 90} }

rule Ciglio .008 {
     SQUARE    {hue 200 sat 0.5}
     Pelo      {r 5 hue 200 sat 0.5}
     Ricciolo  {y -1 s 0.998 b 0.005} }

rule Ricciolo {
     SQUARE    {hue 200 sat 0.5}
     Pelo      {r 5 hue 200 sat 0.5}
     Ricciolo  {y -1 r 3 s 0.998 b 0.005} }

rule Ricciolo .005 {
     SQUARE    {hue 200 sat 0.5}
     Pelo      {r 5 hue 200 sat 0.5}
     Ricciolo  {y -1 r 3 s 0.998 b 0.005 flip 90} }

rule Pelo {
     CIRCLE    {s 5 0.1} }
```

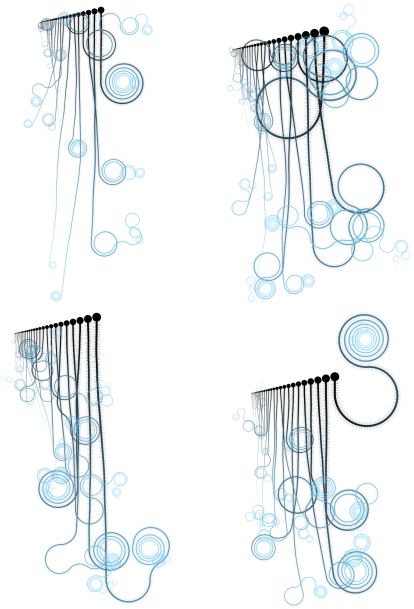


Figure 4.1: On the left, a CFDG adapted from `www.contextfreeart.org/gallery/view.php?id=165`; On the right, different renderings of the left grammar.

- S is the initial symbol.

More specifically, the set of allowed terminals is {SQUARE, TRIANGLE, CIRCLE}. As previously mentioned, CFDGs are augmented grammars[1], where the set of allowed parameters can be grouped into two main groups: (i) colour parameters, such as hue or saturation; (ii) geometric parameters, like size or rotate. More information regarding the parameters' meaning and their allowed values can be found in [11]. Non-terminals can be defined as any string of variable size, requiring that they have to be different from terminals and parameters names. Finally, the initial symbol should be part of the non-terminals set.

Figure 4.1 shows an example of a CFDG along with multiple renderings of it. This is possible due to the fact that CFDGs are non-deterministic grammars, i.e., rendering the same grammar with different rendering seeds possibly leads to visually distinct results. In other words, CFDGs can have the same rule defined more than once. If this happens, when one of those rules is called, one of them is chosen using roulette wheel selection, based on their probabilities. Rule probabilities are defined after their name (see Figure 4.1); if no value is specified, a default of 1 is assumed.

The following subsections are organised as follows. To start, how individuals are mapped from a grammar to an image, as well as the assessment of their quality,

[1]Calls to non-terminals and terminals can take parameters as input.

will be addressed (Section 4.1.1). Later, in Sections 4.1.2 and 4.1.3, tests with both graph and tree-based approaches are going to be described, respectively. Experiments comparing both approaches for the evolution of CFDGs are also to be detailed, in Section 4.1.4.

4.1.1 Fitness Assignment

Context Free Art [29] is an open-source software tool that aims at performing the rendering of CFDGs into images, in this case, into the Portable Network Graphics (PNG) format. In order to deal with the possible recursivity of grammars, which leads to non-terminating programs, the code of Context Free Art was changed to guarantee an output in a maximum number of expansion steps (set by the user).

After the above mentioned adjustment to the software, to produce a single rendering from a grammar it is needed to input to the program the width, height and saving path of the output image along with the input grammar and maximum number of expansion steps. Another important parameter is the rendering seed, which is also part of the rendering's input and assures the replicability of results.

At this step, instead of CFDGs we will have PNG files, from which quality can be measured according to several different metrics. Sections 4.1.1.1 to 4.1.1.5 describe some of these metrics, which will be used during the performance of experiments in the following sections. Then, Section 4.1.1.6 mentions a way of merging multiple metrics into a single one, so that the optimal output should aim at maximising all the characteristics that are sought by each single component.

4.1.1.1 JPEG Size

The image returned by Context Free Art is encoded in JPEG format using the maximum quality settings. The size of the JPEG file becomes the fitness of the individual. The rationale is that, complex images, with abrupt transitions of colour, are harder to compress and hence result in larger file sizes, whereas simple images will result in small file sizes [37, 44]. Although this assignment scheme is rather simplistic, it has the virtue of being straightforward to implement and yielding results that are easily interpretable. As such, it was used to assess the ability of the evolutionary engine to complexify and to establish adequate experimental settings.

4.1.1.2 Number of Contrasting Colours

As the name indicates, the fitness of an individual is equal to the number of contrasting colours present in the image returned by Context Free Art. To calculate the number of contrasting colours we: (i) reduce the number of colours using a quantization algorithm; (ii) sort all colours present in the image by descending order of occurrence; (iii) for all the colours, starting from the most frequent ones, compute the Euclidean distance between the colour and the next one in the ordered list; if it is lower than a certain threshold it is removed from the group; (iv) return as fitness the number of colours present in the list when the procedure is over. In these experiments the Red, Green, Blue (RGB) colour space was adopted. We quantize the image to 256 colours using the quantization algorithm from the Graphics Interchange Format (GIF) format [30]. The threshold was set to 1% of the maximum Euclidean distance between colours ($\sqrt{3 \times 255^2}$ for the RGB colour space).

4.1.1.3 Fractal Dimension and Lacunarity

The use of fractal dimension estimates in the context of computational aesthetic has a significant tradition [51, 75]. Although not as common, lacunarity measures have also been used [7, 8]. For the experiments herein described, the fractal dimension is estimated using the box-counting method and the λ lacunarity value estimated by the sliding box method [31]. By definition, the estimation of the fractal dimension and lacunarity requires identifying the "object" that will be measured. Thus, the estimation methods take as input a binary image (i.e., black and white), where the white pixels define the shape that will be measured and the black pixels represent the background. In our case, the conversion to black and white is based on the CFDG background primitive. All the pixels of the same colour as the one specified by the CFDG background primitive are considered black, and hence part of the background. The ones that are of a different colour are considered part of the foreground (see Figure 4.2). Once the estimates are computed we assign fitness according to the proximity of the measure to a desired value, as follows:

$$fitness = \frac{1}{1 + |target_{value} - observed_{value}|}. \tag{4.1}$$

We use the target values of 1.3 and 0.9 for fractal dimension and lacunarity, respectively. These values were established empirically by calculating the fractal dimension and lacunarity of images that have desirable aesthetic qualities.

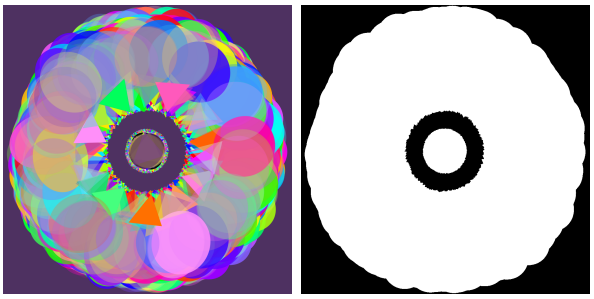


Figure 4.2: Example of the transformation from the input colour image (left image) to the background/foreground image (right image) used for the *Fractal Dimension* and *Lacunarity* estimates.

4.1.1.4 Complexity

This fitness function, based on the work of Machado et al. [37, 43, 44], assesses several characteristics of the image related to complexity. In simple terms, the rationale is valuing images that constitute a complex visual stimulus but that are, nevertheless, easy to process. A thorough discussion of the virtues and limitations of this approach is beyond the scope of this section; as such, we focus on practical issues pertaining its implementation. The approach relies on the notion of compression complexity, which is defined and calculated using the following formula:

$$C(i, scheme) = RMSE(i, scheme(i)) \times \frac{s(scheme(i))}{s(i)}, \tag{4.2}$$

where i is the image being analysed, *scheme* is a lossy image compression scheme, $RMSE$ stands for the root mean square error, and s is the file size function.

To estimate the complexity of the visual stimulus ($IC(i)$) the authors calculate the complexity of the JPEG encoding of the image (i.e., $IC(i) = C(i, JPEG)$). The processing complexity ($PC(i)$) is estimated using a fractal (quadratic tree based) encoding of the image [19]. Considering that as time passes the level of detail in the perception of the image increases, the processing complexity is estimated for different moments in time ($PC(t0, i)$, $PC(t1, i)$) by using fractal image compression with different levels of detail. In addition to valuing images with high visual complexity and low processing complexity, the approach also values images where PC is stable for different levels of detail. In other words, according to this approach, an increase in description length should be accompanied by an increase in image fidelity. Taking all these factors into consideration, Machado et al. [37, 43, 44] proposed the following formula for fitness assignment:

$$\frac{IC(i)^a}{(PC(t0,i) \times PC(t1,i))^b \times \left(\frac{PC(t1,i)-PC(t0,i)}{PC(t1,i)}\right)^c}, \tag{4.3}$$

where a, b and c are parameters to adjust the importance of each component.

Based on previous work by the same authors [43], the ability of the evolutionary engine to exploit the limitations of the complexity estimates was minimised by introducing limits to the different components of the above formula, as follows:

$$\begin{cases} IC(i) & \rightarrow max(0, \alpha - |IC(i) - \alpha|) \\ PC(t0,i) \times PC(t1,i) & \rightarrow \gamma + |(PC(t0,i) \times PC(t1,i)) - \gamma| \\ PC(t1,i) - PC(t0,i) & \rightarrow \delta + |(PC(t1,i) - PC(t0,i)) - \delta|, \end{cases} \tag{4.4}$$

where α, γ and δ operate as target values for $IC(i)$, $PC(t0,i) \times PC(t1,i)$ and $PC(t1,i) - PC(t0,i)$, which were set to 6, 24 and 1.1, respectively. These values were determined empirically through the analysis of images that were considered desirable. Due to the limitations of the adopted fractal image compression scheme, this approach only deals with greyscale images. Therefore, all images have to be converted before processing, as in Figure 4.2.

4.1.1.5 Bell Curve

This fitness function is based on the work of Ross et al. [68] and relies on the observation that many fine-art works exhibit a normal distribution of colour gradients. According to Ross et al. [68], the gradients of each colour channel are calculated, one by one, in the following manner:

$$|\nabla r_{i,j}|^2 = \frac{(r_{i,j} - r_{i+1,j+1})^2 + (r_{i+1,j} - r_{i,j+1})^2}{d^2}, \tag{4.5}$$

where $r_{i,j}$, $g_{i,j}$, $b_{i,j}$ are the image pixel intensity values for position (i,j) for the red, green and blue channels, respectively, and d is a scaling factor that allows the comparison of images of different size; this value was set to 0.1% of half the diagonal of the input image (based on [68]). Then, the overall gradient stimulus, $S_{i,j}$, is computed, as follows:

$$S_{i,j} = \sqrt{|\nabla r_{i,j}|^2 + |\nabla g_{i,j}|^2 + |\nabla b_{i,j}|^2}. \tag{4.6}$$

Next, the response, $R_{i,j}$, is calculated as:

$$R_{i,j} = \log \frac{S_{i,j}}{S_0}, \tag{4.7}$$

where S_0 is a detection threshold (set to 2 as in [68]). After that, the weighted mean (μ) and standard deviation (σ^2) of the response values are determined, as in the following equations:

$$\mu = \frac{\sum_{i,j} {R_{i,j}}^2}{\sum_{i,j} R_{i,j}}, \tag{4.8}$$

$$\sigma^2 = \frac{\sum_{i,j} R_{i,j}(R_{i,j} - \mu)^2}{\sum_{i,j} R_{i,j}}. \tag{4.9}$$

At this point we introduce a subtle, but important change to Ross et al. [68] work; we consider a lower bound for the σ^2, which was empirically set to 0.7. This prevents the evolutionary engine from converging to monochromatic images that, due to the use of a small number of colours, would trivially match a normal distribution. This change has a profound impact on the experimental results, promoting the evolution of colourful images that match a normal distribution of gradients.

Using μ, σ^2 and the values of $R_{i,j}$ a frequency histogram with a bin size of $\sigma/100$ is created, which allows the calculation of the deviation from normality (DFN). The DFN is computed using q_i, which is the observed probability and p_i, the expected probability considering a normal distribution. Ross et al. [68] use:

$$DFN = 1000 \cdot \sum p_i \log \frac{p_i}{q_i}. \tag{4.10}$$

However, based on the results of preliminary runs using this formulation, we found that we consistently obtained better results using:

$$DFN_s = 1000 \cdot \sum (p_i - q_i)^2, \tag{4.11}$$

which measures the squares of the differences between expected and observed probabilities. Therefore, in the experiments described in this section, Bell fitness is assigned according to the following formula:

$$fitness = \frac{1}{1 + DFN_s}. \tag{4.12}$$

4.1.1.6 Combining Different Functions

In addition to the tests where the fitness functions described above were used to guide evolution, we conducted several experiments where the goal was to simultaneously

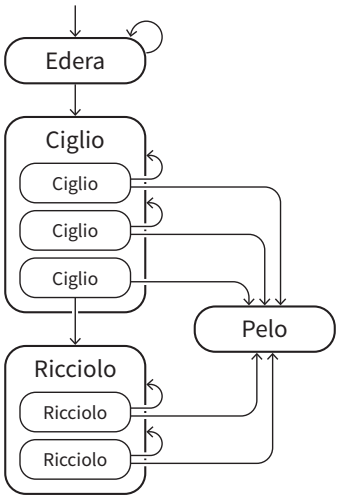


Figure 4.3: Representation of the CFDG of Figure 4.1 as a graph (edges labels, weight of nodes and terminal symbols were omitted for the sake of clarity).

maximise several of these functions. This implied producing a fitness score from multiple functions, which was accomplished using the following formula:

$$combined_{fitness}(i) = \prod_{j} \log\left(1 + f_j(i)\right), \tag{4.13}$$

where i is the image being assessed and f_j refers to the function being considered. Thus, to assign fitness based on the *Complexity* and *Bell* functions we compute: $log(1 + Complexity(i)) \times log(1 + Bell(i))$. By adopting logarithmic scaling and a multiplicative fitness function we wish to promote the discovery of images that maximise all the measures being addressed in the experiment.

4.1.2 Graph-Based

In this approach, individuals are directed graphs, the latter being the object of evolution. An example of a graph capable of depicting a CFDG is given in Figure 4.3, where the grammar introduced in Figure 4.1 is used.

In simple words, the translation between a CFDG and the representation of Figure 4.3 is accomplished, as follows:

1. Create a node for each non-terminal symbol. The node may represent a single production rule (e.g., symbol *Edera* of Figure 4.3) or encapsulate the set of all production rules associated with the non-terminal symbol (e.g., symbols *Ciglio* and *Ricciolo* of Figure 4.3);

Initialisation (see Algorithm 2)	Values
min, max number of symbols	(1,3)
min, max number of rules	(1,3)
min, max calls per production rule	(1,2)
Evolutionary Engine	**Values**
Number of runs	30
Number of generations	100
Population size	100
Crossover probability	0.6
Mutation probability	0.1
Tournament size	10
Elite size	Top 2% of the population
CFDG Parameters	**Values**
Maximum number expansion steps	100000
Limits of the geometric transformations	rotate ∈ [0,359], size ∈ [-5,5] x ∈ [-5,5], y ∈ [-5,5], z ∈ [-5,5] flip ∈ [-5,5], skew ∈ [-5,5]
Limits of the colour transformations	hue ∈ [0,359], saturation ∈ [-1,1] brightness ∈ [-1,1], alpha ∈ [-1,1]
Terminal symbols	SQUARE, CIRCLE, TRIANGLE

Table 4.1: Parameters used for the graph-based approach experiments.

2. Create edges between each node and the nodes corresponding to the terminals and non-terminals appearing in its production rules;

3. Annotate each edge with the corresponding parameters (e.g., in Figure 4.3 the edges to *Pelo* possess the label '{r 5 hue 200 sat 0.5}').

Experiments using the above mentioned form of representing individuals and the operators suit to it (described in Section 3.1) will be detailed in the upcoming subsections. First, in Section 4.1.2.1, we will specify the experimental setup. Next, focus is given to aspects regarding the graph-topology of graphs (Section 4.1.2.2) and the non-determinism of grammars (Section 4.1.2.3). To end, experiments featuring the application of the previously presented fitness assignment schemes are performed, in Sections 4.1.2.4 and 4.1.2.5.

The work herein described led to the publication of a book chapter [40] (see Appendix E.1), on which the following subsections are partially based. As part of a deliverable to ConCreTe EU Funded Project a web-interface for the performance of experiments and the visualisation of the results had also to be developed (Appendix B). Data from this section was used in a publication focusing ways to enhance the visualisation of this type of representations in evolutionary algorithms (Appendix E.2).

4.1.2.1 Experimental Setup

In order to assess the adequacy of the evolutionary engine for the evolution of CFDGs and to determine a reasonable set of configuration parameters, several tests using *JPEG Size* as fitness function were performed. The rationale behind the use of this metric is just the ease of analysing the results. Table 4.1 summarises the experimental parameters that are going to be used throughout all the experiments herein described.

One can argue that the presented parameters are not the optimal ones, due to the fact that only one fitness function is being used to propose them. This is correct; although, during tests, results showed that the engine is not overly sensitive. The search for an optimal parametrisation set is then considered out of the scope of this Dissertation, aiming instead at determining the capacity of the system to properly evolve CFDGs.

4.1.2.2 Graph Topology

An aspect directly related with the graph representation of individuals is the possibility of existence of unconnected nodes[2]. This was also a target of study and, in order to have better insight on its consequences, three setups were tested:

Unrestricted – crossover points are randomly chosen;

Restricted – crossover points are randomly chosen from the list of reachable nodes of each parent;

Restricted with Cleaning – in addition to enforcing the crossover to occur in a reachable region of the graph, after applying crossover and mutation all unreachable nodes are deleted.

Figure 4.4 displays the evolution of the best and average fitnesses across all generations and runs of the above setups. As can be observed, although the behaviours of the three different approaches are similar, the restricted versions consistently outperform the unrestricted implementation by a small, yet statistically significant, margin. The differences between the restricted approaches are not statistically significant.

The differences among the three approaches become more visible when we consider the evolution of the number of reachable and unreachable nodes through time. As it can be observed in Figure 4.5, without cleaning, the number of unreachable nodes grows significantly, clearly outnumbering the number of reachable nodes. The number

[2]Nodes that are not reachable from the startshape.

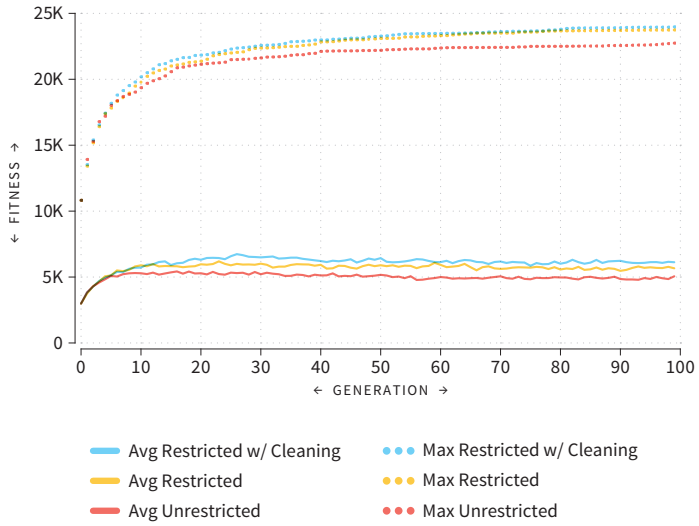


Figure 4.4: Best and average fitness values for different implementations of the genetic operators using *JPEG Size* as fitness function. Results are averages of 30 independent runs.

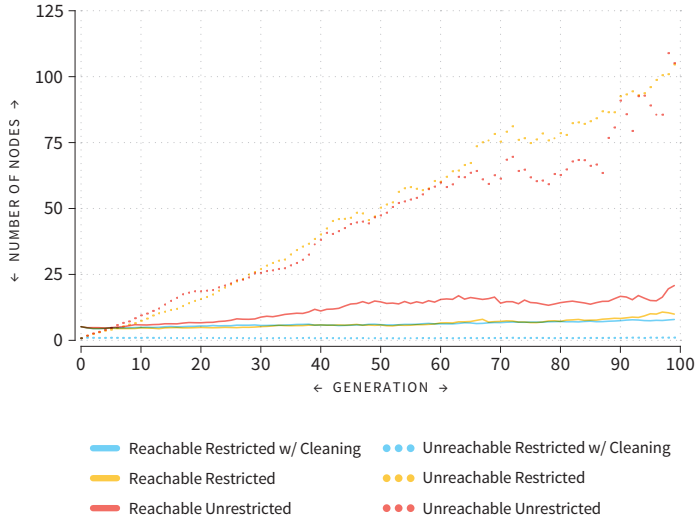


Figure 4.5: Evolution of the average number of reachable and unreachable nodes across generations for different implementations of the genetic operators using *JPEG Size* as fitness function. Results are averages of 30 independent runs.

of reachable nodes of the restricted versions is similar, and smaller than the one resulting from the unrestricted version. Although cleaning does not significantly improve fitness in comparison with the restricted version, the reduction of the number of rules implies a reduction of the computational cost of interpreting the CFDGs and

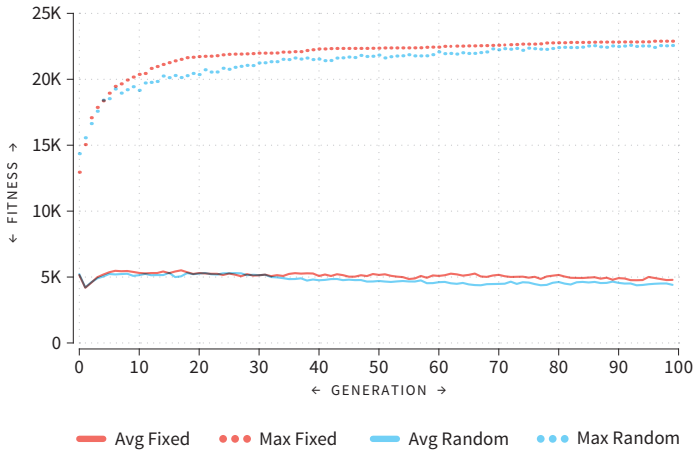


Figure 4.6: Evolution of the best and average fitness across generations when using fixed and random rendering seeds using *JPEG Size* as the fitness function. Results are averages of 30 independent runs.

applying the crossover operator. As such, taking these experimental findings into consideration, we adopt the *Restricted with Cleaning* variant in all further tests.

4.1.2.3 Non-Determinism

The non-deterministic nature of the CFDGs implies that each genotype may be mapped into a multitude of phenotypes (see Figure 4.1). The genotype to phenotype mapping of a non-deterministic grammar depends on a rendering seed, which is passed to Context Free Art. We considered two scenarios: (i) using a fixed rendering seed for all individuals; (ii) randomly generating the rendering seed whenever the genotype to phenotype mapping occurs. The second option implies that the fitness of a genotype may, and often does, vary from one generation to the next, since its phenotype may change.

Figure 4.6 summarises the results of these tests in terms of the evolution of fitness through time. As expected, using a fixed rendering seed yields better fitness, but the difference between the approaches is surprisingly small and decreases as the number of generations increases. To better understand this result we focused on the analysis of the characteristics of the CFGDs being evolved. Figure 4.7 depicts box plots of fitness values of the fittest individuals of each of the 30 evolutionary runs using different setups:

Fixed – individuals evolved and evaluated using fixed rendering seeds;

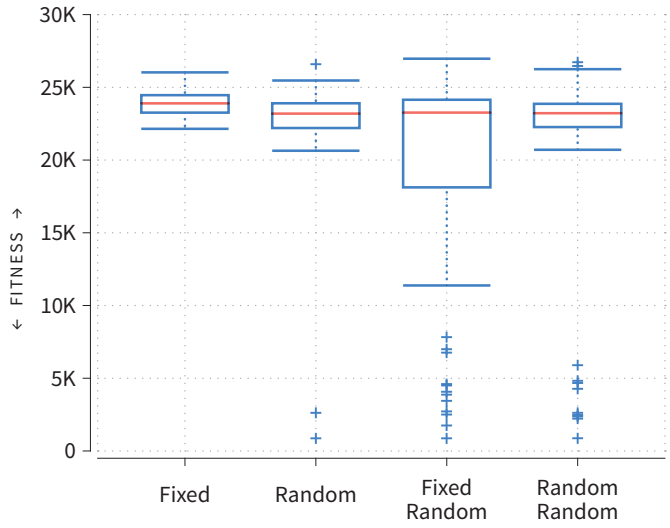


Figure 4.7: Box plots of fitness values of the fittest individuals of each of the 30 evolutionary runs using different rendering seed setups.

Random – individuals evolved using random rendering seeds and evaluated using the same seeds as the ones picked randomly during evolution;

Fixed Random – individuals evolved using fixed rendering seeds and evaluated with 30 random seeds each;

Random Random – individuals evolved using random rendering seeds and evaluated with 30 random seeds each.

In other words, we take the genotypes evolved in a controlled static environment (fixed random seed) and place them in different environments, proceeding in the same way for the ones evolved in a changing environment. The analysis of the box plots shows that, in the considered experimental settings, the fitness of the individuals evolved in a fixed environment may change dramatically when the environmental conditions are different. Conversely, using a dynamic environment promotes the discovery of robust individuals that perform well under different conditions. Although this result is not unexpected, it was surprising to notice how fast the evolutionary algorithm was able to adapt to the changing conditions and find robust individuals. In future tests we wish to explore, and exploit, this ability. Nevertheless, for the purposes of this section, and considering that the use of a fixed rendering seed makes the analysis and reproduction of the experimental results easier, we adopt a fixed rendering seed in all further tests.

4.1.2.4 Single Fitness Guiding

We will now focus on the evolution of CFDGs, using the previously defined fitness functions. Figure 4.8 summarises the results of these experiments in terms of evolution of fitness. Each chart depicts the evolution of the fitness of the best individual when using the corresponding fitness function to guide evolution. The values yielded by the other 5 fitness functions are also depicted for reference to illustrate potential inter-dependencies among fitness functions. The values presented in each chart are averages of 30 independent runs (180 runs in total). To improve readability we have normalised all the values by dividing each raw fitness value by the maximum value for that fitness component found throughout all the runs.

The most striking observation pertains the *Fractal Dimension* and *Lacunarity* fitness functions. As it can be observed, the target values of 1.3 and 0.9 are easily approximated even when these measures are not used to guide fitness. Although this is a disappointing result, it is an expected one. Estimating the fractal dimension (or lacunarity) of an object that is not a fractal and that can be described using Euclidean geometry yields meaningless results. That is, even though you obtain a value, this value is meaningless in the sense that there is no fractal dimension to be measured. As such, these measures may fail to capture any relevant characteristic of the images. In the considered experimental conditions, the evolutionary algorithm was always able to find, with little effort, non-fractal images that yield values close to the target ones. Most often than not, these images are rather simplistic. We conducted several tests using different target values, obtaining similar results.

An analysis of the results depicted in Figure 4.8 reveals that maximising *JPEG Size* promotes *Contrasting Colours* and *Complexity*, but does not promote a distributing of gradients approaching a normal distribution (*Bell*). Likewise, maximising *Contrasting Colours* originates an improvement in *JPEG Size* and *Complexity* during the early stages of the evolutionary process; *Bell* is mostly unaffected. Using *Complexity* to guide evolution results in an increase of *JPEG Size* and *Contrasting Colours* during the early stages of the runs, but the number of *Contrasting Colours* tends to decrease as the number of generations progresses. The *Complexity* fitness function operates on a greyscale version of the images, as such, it is not sensitive to changes of colour. Furthermore, abrupt changes from black to white create artefacts that are hard to encode using JPEG compression, resulting in high IC estimates. Fractal image compression, which is used to estimate PC, is less sensitive to these abrupt changes. Therefore, since the approach values images with high IC and low PC, and

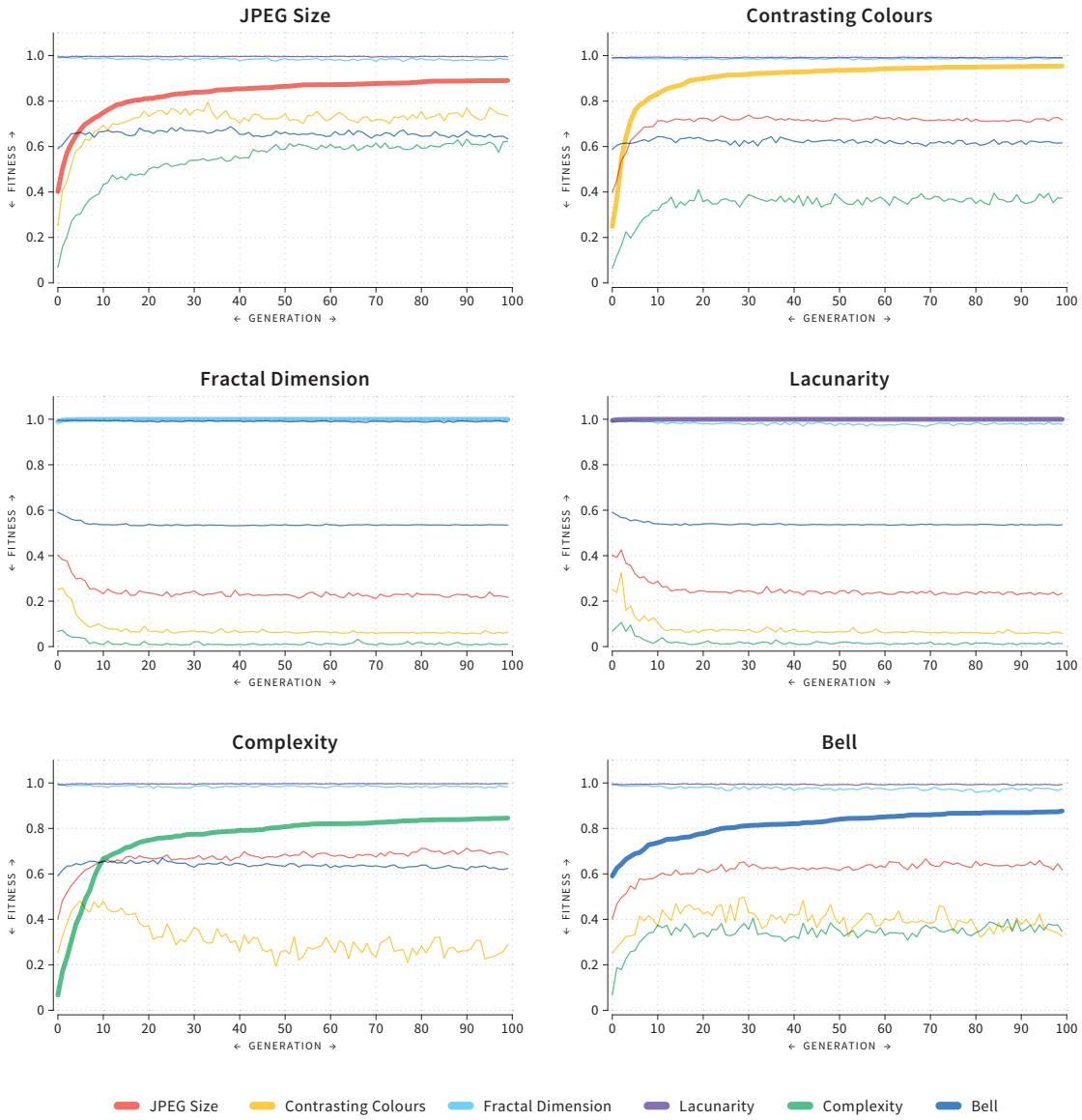
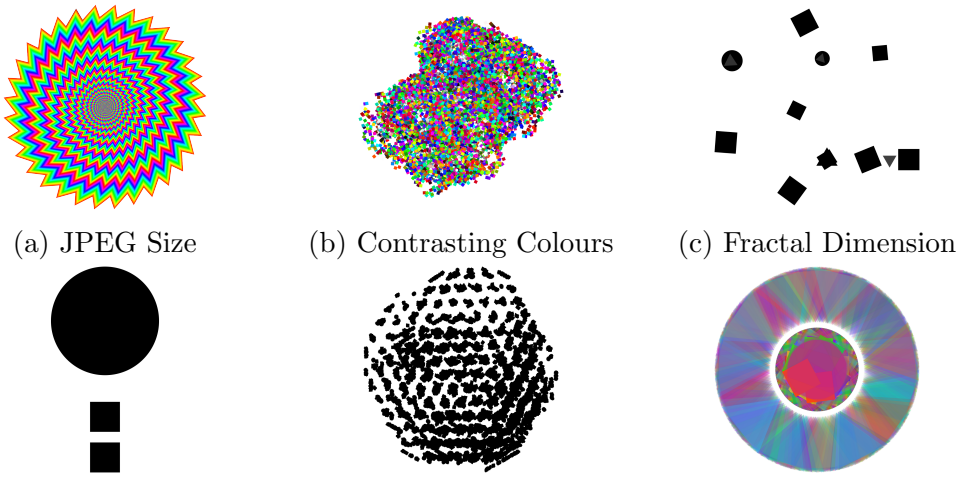


Figure 4.8: Evolution of the fitness of the best individual across generations. The fitness function used to guide evolution is depicted in the title of each chart and is represented with a thicker line. The other values are presented for reference. Results are averages of 30 independent runs for each chart and have have been normalised to improve readability.

since it does not take colour information into consideration, the convergence to images using a reduced palette of contrasting colours is expected. Like for the other measures, *Complexity* and *Bell* appear to be unrelated. Finally, maximising *Bell* promotes an increase of *JPEG Size*, *Contrasting Colours* and *Complexity* during the first generations. It is important to notice that this behaviour was only observed after



(a) JPEG Size (b) Contrasting Colours (c) Fractal Dimension

(d) Lacunarity (e) Complexity (f) Bell

Figure 4.9: Examples of individuals evolved using each one of the fitness functions.

enforcing a lower bound for σ^2 (see Section 4.1.1.5). Without this limit, maximising *Bell* results in the early convergence to simplistic monochromatic images (typically a single black square on a white background). The adoption of a quadratic deviation from normality estimate (DFN_s) also contributed to the improvement of the visual results.

Sample individuals of each fitness function can be seen in Figure 4.9. More examples are available in Appendix E.1. As expected, *JPEG Size* tends to converge to colourful circular patterns, with high contrasts of colours. The tendency to converge to circular patterns, which is observed in several runs, is related with the recursive nature of the CFDGs and the particularities of the Context Free rendering engine. For instance, repeatedly drawing and rotating a square while changing its colour will generate images that are hard to encode. Furthermore, the rendering engine automatically "zooms in" the shapes drawn, cropping the empty regions of the canvas. As such, rotating about a fixed point in space tends to result in images that fill the canvas, maximising the opportunities for introducing abrupt changes and, therefore, maximising file size.

Contrasting Colours, as predictable, converges to images that are extremely colourful. *Complexity* tends to promote convergence to monochromatic and highly structured images. Moreover, since fractal image compression takes advantage of the self-similarities present in the image at multiple scales, the convergence to structured and self-similar structures that characterises these runs was expected.

Images evolved using *Bell* depict a structured variation of colour, which is easily explained by the need to match a natural distribution of colour gradients.

Figure 4.10: Example of the rendering of the best individual from the initial (left image) and last (right image) population, using *Fractal Dimension* and *Contrasting Colours* as fitness function. As observed, despite depicting a high number of colours, no fractal pattern is identifiable.

Finally, using *Fractal Dimension* and *Lacunarity* leads to uninteresting results, as explained earlier in this section.

4.1.2.5 Multiple Fitness Guiding

We performed several experiments where a combination of measures was used to assign fitness (see Section 4.1.1.6). We conducted tests combining *Fractal Dimension* and *Lacunarity* with other measures. Results confirm that these metrics are ill-suited for aesthetic evolution in the considered experimental settings (see Figure 4.10). Tests using *JPEG Size* in combination with other measures were also performed. The analysis of the results indicates that they are subsumed and surpassed by those obtained when using *Complexity* in conjunction with other metrics. This results from two factors: on one hand *Complexity* already takes into account the size of the JPEG encoding; on the other, the limitations of *Complexity* regarding colour are compensated by the use of measures that are specifically designed to handle colour information. As such, taking into account the results described in the previous section, we focus on the analysis of the results obtained when combining: *Contrasting Colours*, *Complexity* and *Bell*.

Figure 4.11 summarises the results of these experiments in terms of evolution of fitness. Each chart depicts the evolution of the fitness of the best individual when using the corresponding combination of measures as fitness function. The values yielded by the remaining measures are depicted but do not influence evolution. The values presented in each chart are averages of 30 independent runs (120 runs in total). As previously, the values have been normalised by dividing each raw fitness value by the maximum value for that fitness component found throughout all the runs.

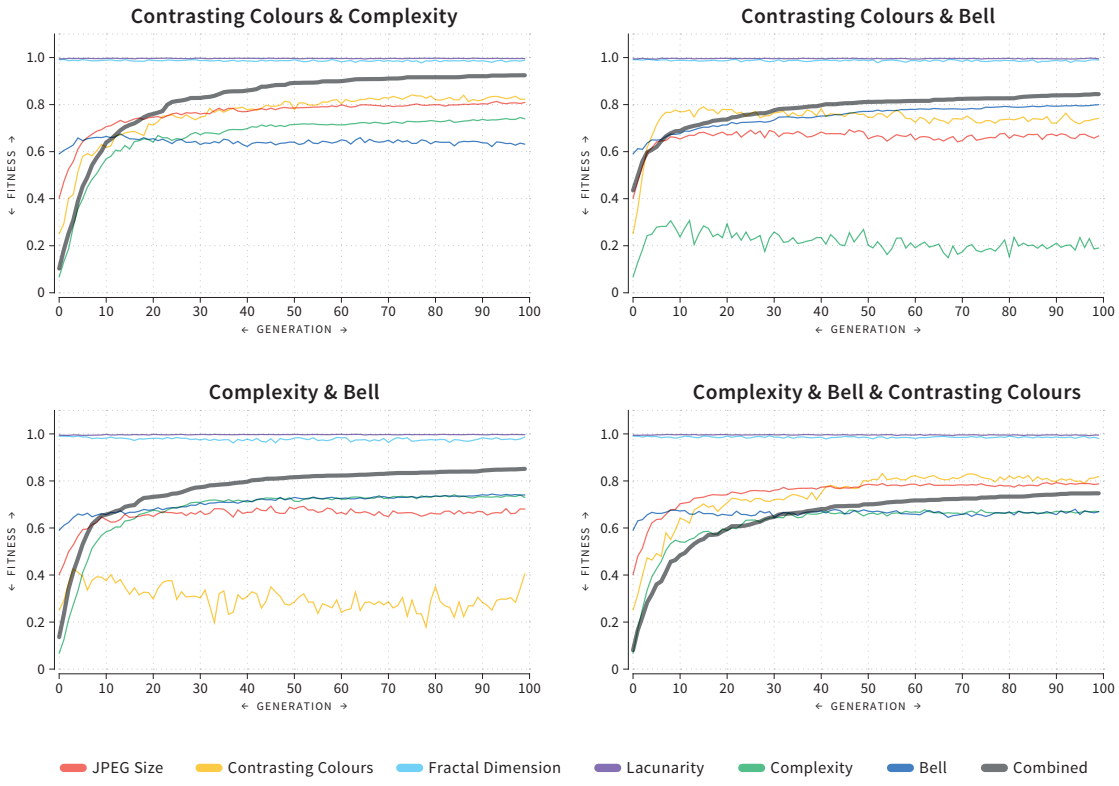


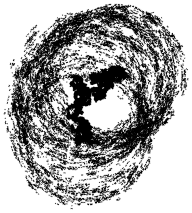
Figure 4.11: Evolution of the fitness of the best individual across generations using a combination of measures. The combination used to guide evolution is depicted in the title of each chart. The other values are presented for reference, but have no influence in the evolutionary process. Results are averages of 30 independent runs for each chart and have have been normalised to improve readability.

As can be observed, combining *Contrasting Colours* and *Complexity* leads to a fast increase of both measures during the early stages of the runs, followed by a steady increase of both components throughout the rest of the runs. This shows that, although the runs using *Complexity* alone converged to monochromatic imagery, it is possible to evolve colourful images that also satisfy the *Complexity* measure.

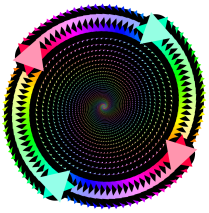
Combining *Contrasting Colours* and *Bell* results in a rapid increase of the number of contrasting colours during the first generations. Afterwards, increases in fitness are mainly accomplished through the improvements of the *Bell* component of the fitness function. This indicates that it is easier to maximise the number of contrasting colours than to attain a normal distribution of gradients. This observation is further attested by the analysis of the charts pertaining the evolution of fitness when using *Contrasting Colours*, *Complexity* and *Bell* individually, which indicates that *Bell* may be the hardest measure to address. The combination of *Complexity* and *Bell* is characterised

(a) Contrasting Colours & Complexity

(b) Contrasting Colours & Bell



(c) Complexity & Bell



(d) Complexity & Bell & Contrasting Colours

Figure 4.12: Examples of individuals evolved using combined metrics as fitness function.

by a rapid increase of complexity during the first populations, followed by a slow, but steady, increase of both measures throughout the runs. The combination of the three measures further establishes *Bell* as the measure that is most difficult to address, since the improvements of fitness are mostly due to increases in the other two measures.

Similarly to the last section, sample images obtained during the performance of the previously described tests can be seen in Figure 4.12. More examples are available in Appendix E.1. Combining *Contrasting Colours* and *Complexity* led to individuals that depict not only the structures already seen while using *Complexity* but that are also colourful, instead of monochromatic. Analogously for the scenario where *Contrasting Colours* and *Bell* are used as one fitness function. In this case, images tend to show a normal distribution of the colour's gradient but, when comparing to the ones only evolved with *Bell* they outstand for being more colourful.

To conclude, using *Complexity* and *Bell* produces images that have more colours than those output using only *Complexity*, although they are not as colourful as expected. For that, *Contrasting Colours* was added to the combined fitness function, increasing, as predicted, the number of colours.

4.1.3 Tree-Based

As previously mentioned, in Section 3.2, in the tree-based approach individuals are represented as derivation trees. In order to accomplish that, the user must first input

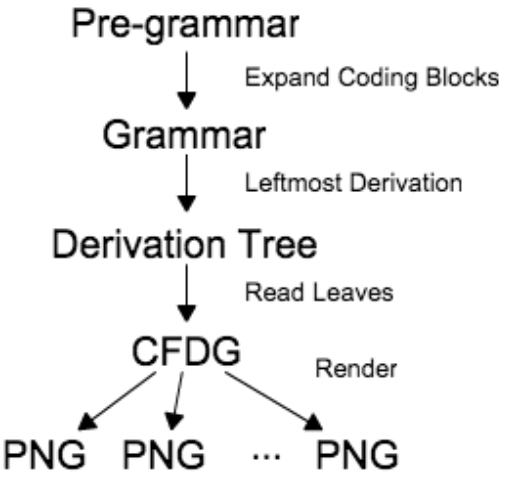


Figure 4.13: Pre-grammars flow chart adapted to the CFDGs scenario.

the pre-grammar, as a BNF with the addition of special coding blocks, which specifies how the allowed derivation steps should be formed (Section 3.2.1). Figure 4.13 adapts the flow chart of Section 3.2.1 to the domain of CFDGs. First, a pre-grammar specifying the structure that the grammar that is able to produce CFDGs should follow has to be provided. Then, from this grammar, derivation trees for CFDGs are built. However, CFDGs are also grammars and, as so, we do in fact have three grammatical levels (pre-grammar – expanded pre-grammar – CFDG). From this last level, it is later possible to generate PNGs and in this way assess the quality of the CFDG. Evolution acts directly over the derivation tree and consequently, indirectly over the CFDG.

For the current application scenario the used pre-grammar can be seen in Grammar 4.1, where the first rule to be expanded is always the first production rule of the input pre-grammar, in this case, <S>.

A brief analysis of the pre-grammar shown in Grammar 4.1 allows some conclusions regarding the composition of the derivation trees it generates. Beginning from the start symbol (<S>), every generated derivation tree is composed of subtrees referring to the startshape, background and a variable number of rules (from 4 to 6). As the reader may remember, CFDGs are non-deterministic grammars. That is, there are non-terminal rules which can be defined multiple times and then, while rendering, one of the possibilities is chosen every time the non-terminal is called. This property is assured by the use of the special blocks, previously detailed in Section 3.2.1. The first pre-rule of Grammar 4.1 initially generates each of the 3 non-terminal symbols (<RULE>(3, NONTERMINAL, STRING, True)) and then repeats some of them (<RULE>(1,3)) a random number of times (between 1 and 3).

```
<S> ::= <STARTSHAPE> <BACKGROUND>
        <RULE>(3, NONTERMINAL, STRING, True) <RULE>(1,3)
<STARTSHAPE> ::= startshape <NONTERMINAL>
<BACKGROUND> ::= background { <COLORPARAMETERS> }
               |  λ
<RULE> ::= rule <NONTERMINAL> <PROB> { <CALLS> }
<COLORPARAMETERS> ::= <COLORPARAMETER> <COLORPARAMETERS>
                    | λ
<COLORPARAMETER> ::= brightness [-1, 1]
                   |  hue [0, 359]
                   |  saturation [-1, 1]
                   |  alpha [-1, 1]
<PROB> ::= [0,1]
         |  λ
<CALLS> ::= <SYMBOL> { <PARAMETERS> } <CALLS>
          |  λ
<SYMBOL> ::= <TERMINAL>
           |  <NONTERMINAL>
<PARAMETERS> ::= <COLORPARAMETER> <PARAMETERS>
               |  <GEOMETRICPARAMETER> <PARAMETERS>
               |  λ
<TERMINAL> ::= SQUARE
             |  CIRCLE
             |  TRIANGLE
<GEOMETRICPARAMETER> ::= x [-5, 5]
                       |  y [-5, 5]
                       |  skew [-5, 5] [-5, 5]
                       |  z [-5, 5]
                       |  flip [-5, 5]
                       |  rotate [0, 359]
                       |  size [-5, 5]
```

Grammar 4.1: Pre-grammar used for the evolution of CFDGs where the number of <RULE> subtrees is established upon population initialisation.

In the following subsections we will present the experimental setup used for the tests herein described (Section 4.1.3.1), moving then to detailing the performed experiments (Sections 4.1.3.2, 4.1.3.3 and 4.1.3.4), most of them similar to the ones done in the previous section, with the graph-based approach.

Evolutionary Engine	Values
Number of runs	30
Number of generations	100
Population size	100
Crossover probability	0.6
Mutation probability	0.3
Leaves probability	0.2
Tournament size	10
Elite size	Top 2% of the population
CFDG Parameters	**Values**
Maximum number expansion steps	100000
Limits of the geometric transformations	Check used pre-grammar
Limits of the colour transformations	Check used pre-grammar
Terminal symbols	Check used pre-grammar

Table 4.2: Parameters used for the tree-based approach experiments.

4.1.3.1 Experimental Setup

This section aims at detailing the setup used to assess the adequacy of the system to properly evolve CFDGs. Table 4.2 contains the parameters used during the following experiments. As in Section 4.1.2.1, they were reached after running preliminary tests with the *JPEG Size* fitness function and are, almost certainly, not the optimal ones for all fitness functions.

```
<S> ::= <STARTSHAPE> <BACKGROUND>
        <RULE>(3, NONTERMINAL, STRING, True) <OTHERRULES>
<OTHERRULES> ::= <RULE> <OTHERRULES>
               |  λ

...
```

Grammar 4.2: Pre-grammar used for the evolution of CFDGs where the number of <RULE> subtrees is dynamic, through the application of genetic operators. For simplicity reasons only the rules that differ from Grammar 4.1 were detailed.

4.1.3.2 Pre-grammar Formulation

As one may have noticed, the previous introduced pre-grammar (Grammar 4.1) forces individuals to follow a restricted structure, at least in what concerns the number of <RULE> subtrees, which are created upon individual initialisation. For that, another pre-grammar formulation was tested, in order to determine if the limitation of the number of allowed rules constrains the capacity of the engine to evolve the desired results. Grammar 4.2 raises this limitation, by replacing, in the production rule S, the

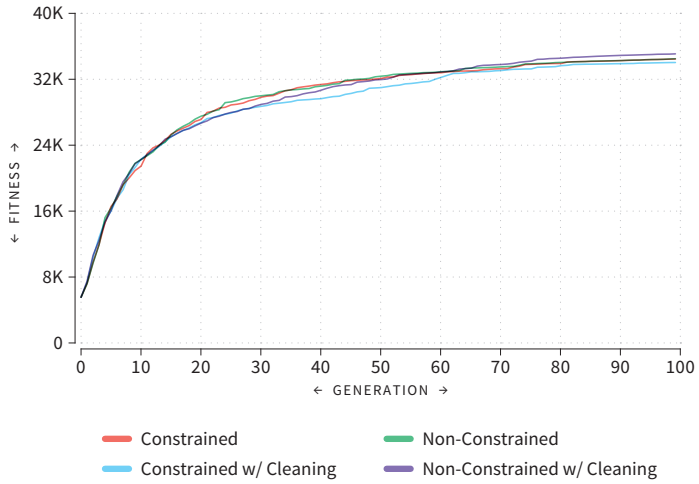


Figure 4.14: Evolution of the fitness of the best individual across generations for the different setups. Results are averages of 30 independent runs.

block <RULE>(1,3) by the non-terminal <OTHERRULES>, allowing the individual to grow or decrease in the number of <RULE> subtrees when mutation or crossover are applied to him.

Moreover, we are also going to address the unreachable rules situation. In the context of a tree representation we can state that this problem variable is different then when using graphs, because unreachable rules of the CFDG are not explicitly present in a tree. Although, when thinking about the CFDGs structure, it is easy to identify that not all <RULE> subtrees will be called from <CALL> subtrees. In line with the previous, we developed a method for mapping a CFDG derivation tree into the representation used in the graph approach, and vice-versa. Consequently, after applying the mapping from a tree to a graph, we erase all the unreachable rules subtrees, reconverting then the graph back to a derivation tree.

From the above, we designed four tests, addressing the following experimental conditions:

Constrained – using Grammar 4.1, which does not allow the number of <RULE> subtrees to change;

Non-Constrained – using Grammar 4.2, that has the possibility of increasing or decreasing the number of <RULE> subtrees, which is facilitated by the <OTHERRULES> derivation step;

Constrained with Cleaning – based on the constrained testing scenario and, additionally, erases all the unreachable parts of the CFDGs;

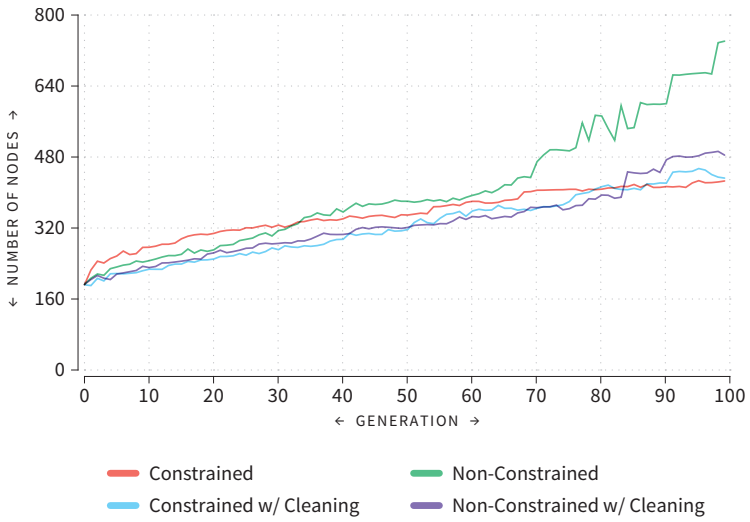


Figure 4.15: Evolution of the number of tree nodes across generations, for the different setups. Results are averages of 30 independent runs.

Non-Constrained with Cleaning – based on the non-constrained testing scenario and, additionally, erases all the unreachable parts of the CFDGs.

Figure 4.14 shows the evolution of the fitness of the best individual over 30 independent runs. Care was taken so that both tests start with the same initial population, in order to facilitate comparison. As depicted, regardless of the experimental conditions, all setups converge similarly, attaining the same results.

Although, if instead of fitness evolution, we look at the number of nodes (internal and leaves) in the derivation tree, Figure 4.15, as expected, the non-constrained approach leads to the higher number of nodes. All the remaining setups have similar results, being in the first generations the number of nodes in the constrained one a bit superior to both approaches with cleaning. However, approximately from generation 80 onwards both setups using the constrained pre-grammar yield very similar results, with less nodes than the non-constrained with cleaning.

From the stated in the previous paragraphs, in the remaining tests we adopt the *Constrained* setup. We opt for not using cleaning because a subtree, even if not meaningful to one individual, upon crossover it can become relevant to another one. The *Non-Constrained* approach is not chosen because, despite the similarity in fitness, an higher number of nodes implies a more costly level of computation.

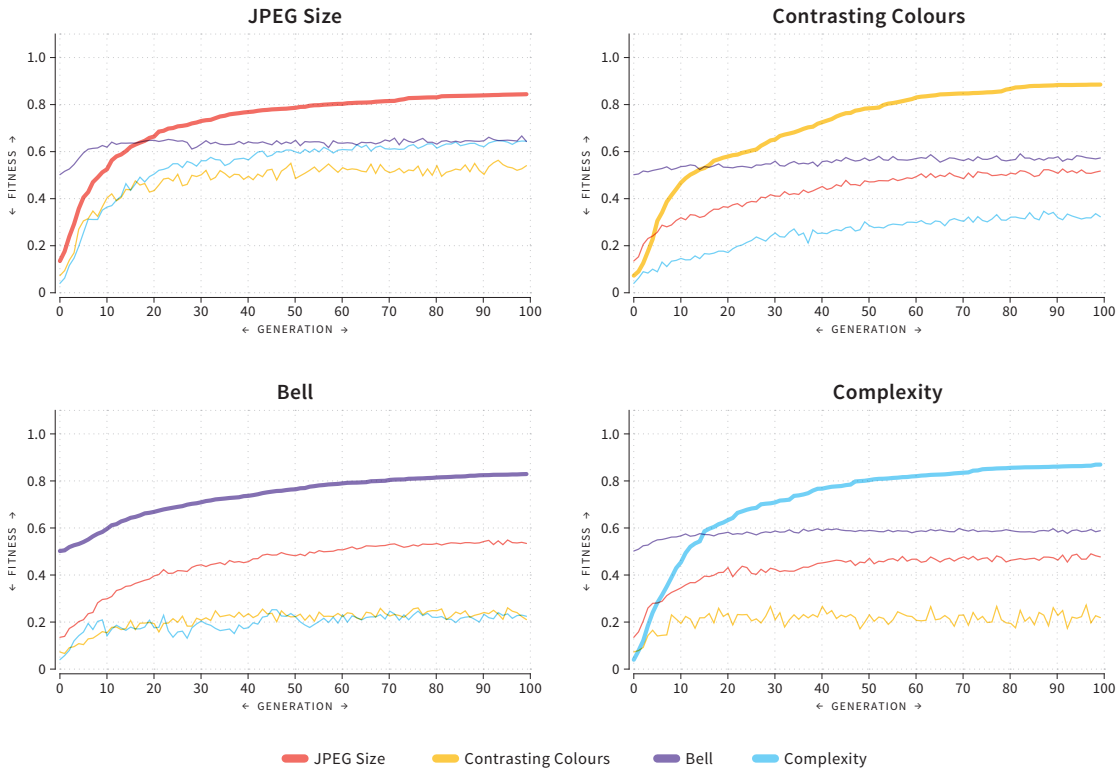
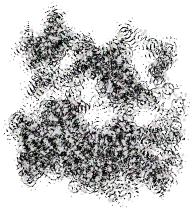


Figure 4.16: Evolution of the fitness of the best individual across generations. The fitness function used to guide evolution is depicted in the tittle of each chart and is represented with a thicker line. The other values are presented for reference, but have no influence in the evolutionary process. Results are averages of 30 independent runs for each chart and have been normalised to improve readability.

4.1.3.3 Single Fitness Guiding

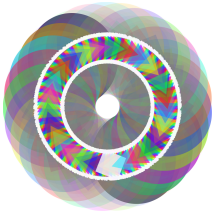
Taking into account the experiments performed with only one guiding fitness and the graph-based approach (Section 4.1.2.4), we will only design tests using as fitness function the *JPEG Size*, *Contrasting Colours*, *Bell* and *Complexity*. *Fractal Dimension* and *Lacunarity* are not going to be addressed due to their lack of capacity to promote the evolution of non-fractal images into fractals.

Figure 4.16 depicts the results of these experiments in terms of fitness evolution. Each chart summarises the evolution of the fitness of the best individual when using the corresponding fitness function to guide evolution. The values yielded by the other fitness functions are also shown to illustrate possible inter-dependencies. Moreover, the values presented in each chart are averages of 30 independent runs (120 runs in total). In order to improve readability the values have been normalised, by dividing the raw value by the maximum one for that fitness component found in all performed



(a) JPEG Size

(b) Contrasting Colours



(c) Bell

(d) Complexity

Figure 4.17: Examples of individuals evolved using each one of the fitness functions.

runs.

An analysis of the results leads to conclusions similar to those of Section 4.1.2.4, where the same tests were performed with the graph-based approach. Maximising *JPEG Size* promotes *Contrasting Colours* and *Complexity*, but not *Bell*, which only has a little increase during the first 10 generations and then stabilises. The same can be stated about *Contrasting Colours*; maximising it conducts to enhancements in *JPEG Size* and *Complexity*. *Bell* is unaffected, meaning that despite the increase in the number of colours present in images they do not have a normal distribution of colour gradients, resembling instead random patterns. Maximising *Bell* boosts *JPEG Size* and *Contrasting Colours*, the latter at a slower rate; *Complexity* does not seem to be related, as its behaviour is erratic during the course of the experiment. Finally, maximising *Complexity* is accompanied by a steady increase in *JPEG Size* and *Contrasting Colours*, at least during the first generations. The number of contrasting colours then stabilises because the *Complexity* metric operates over greyscale images and, as such, colours are not taken into consideration. *Bell* appears to be uncorrelated.

Sample individuals of each fitness function can be found in Figure 4.17. More examples are available in Appendix C. As expected, the results confirm the predictable, being the images similar to those evolved with the graph-based approach and, as such, the conclusions the same. *JPEG Size* tends to converge to high contrast colourful patterns. *Contrasting Colours*, results in images that are extremely colourful, where no structure in the colours placement seems to exist. On the other hand, *Bell* depicts

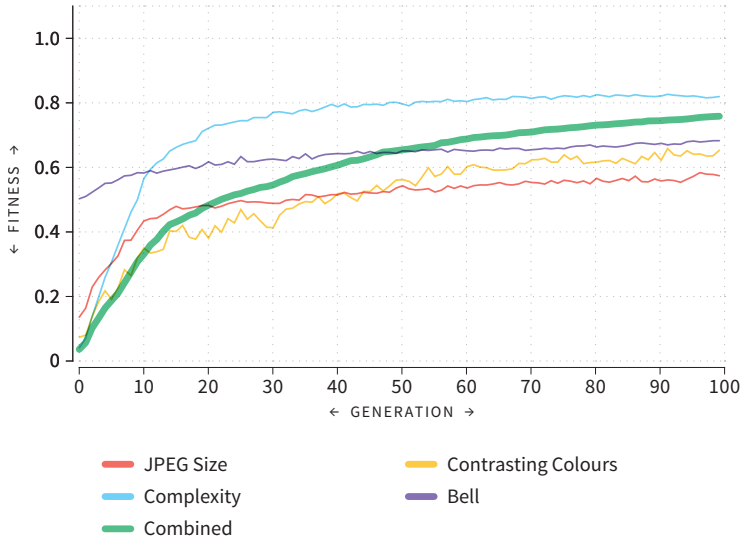


Figure 4.18: Evolution of the fitness of the best individual across generations using as fitness function a combination of *Complexity*, *Bell* and *Contrasting Colours* (thicker line). The other values are presented for reference, but have no influence in the evolutionary process. Results are averages of 30 independent runs for each chart and have been normalised to improve readability.

a structured variation of colour, explained after the need to match a normal distribution of colour gradients. Finally, using *Complexity* generates structured images, most of them monochromatic, because of the need to convert them to greyscale before assessing fitness.

4.1.3.4 Multiple Fitness Guiding

As previously done with the graph-based approach, Section 4.1.2.5, experiments where the fitness function acts as a combination of others were also performed. The aim of the present section is to prove the capacity of the approach to deal with different objectives during evolution, as such, only one test will be presented, where the combination of *Complexity*, *Bell* and *Contrasting Colours* is sought. *JPEG Size* was not used because *Complexity* already takes it into consideration.

Figure 4.18 shows the results of the experiment in what regards fitness evolution, as a combined measure, and individually. An example of a produced individual is depicted in Figure 4.19 (for further examples check Appendix C).

The combination of *Complexity*, *Bell* and *Contrasting Colours* is characterised by a fast increase in the *Complexity* component during the first 40 generations, which

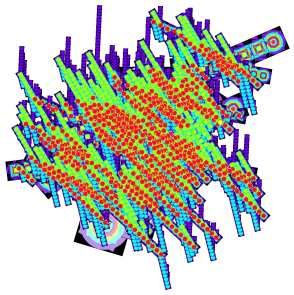


Figure 4.19: Example of one individual evolved using *Complexity*, *Bell* and *Contrasting Colours* as fitness function.

is followed by a steady evolution of the number of *Contrasting Colours*. *Bell* can be identified as the most difficult metric to address, since the improvements in the combined fitness function are mostly result of the evolution of the two other measures. *JPEG Size*, despite not being treated in the combined guiding function, is also improving, which is expected because the *Complexity* metric considers it.

Visually, results often depict characteristics that can be identified with each one of the three measures. That is, highly structured images, with a great number of contrasting colours, usually depicting a normal distribution of gradients.

4.1.4 Merging Both Approaches

After proving, in Sections 4.1.2 and 4.1.3, that both graph and tree-based approaches are capable of promoting evolution, in this case, of CFDGs, we wonder if combining distinct parts from each one may or not lead to better results. This new form of addressing the problem is going to be tackled in the next paragraphs.

To initialise the population that will seed the evolutionary process we opt for the one used in the tree-based approach, i.e., the creation of random derivation trees based on an a-priori defined pre-grammar. This choice was made with regard to which initialisation procedure is most easily generalisable.

The question now turns to which combination of crossover and mutation operators to use. For that, tests using all the possibilities are going to be performed in the following subsections. In order to assure the validity of the solutions we use a method that maps derivation trees into graphs, and vice-versa. This procedure is used before the application of the graph-crossover and graph-mutation operators to map from a derivation tree to graph and then, after the operators application, to map the individuals back to a derivation tree.

Evolutionary Engine	Values
Number of runs	30
Number of generations	300
Population size	100
Crossover probability	0.6
Mutation probability	0.3
Tournament size	10
Elite size	Top 2% of the population
CFDG Parameters	**Values**
Maximum number expansion steps	100000
Limits of the geometric transformations	rotate ∈ [0,359], size ∈ [-5,5] x ∈ [-5,5], y ∈ [-5,5], z ∈ [-5,5] flip ∈ [-5,5], skew ∈ [-5,5]
Limits of the colour transformations	hue ∈ [0,359], saturation ∈ [-1,1] brightness ∈ [-1,1], alpha ∈ [-1,1]
Terminal symbols	SQUARE, CIRCLE, TRIANGLE
Other Parameters	**Values**
Graph per-gene mutation probability	0.1
(Tree, Graph) combined crossover probabilities	$\{(0.3, 0.7), (0.5, 0.5), (0.3, 0.7)\}$

Table 4.3: Parameters used for the experiments merging graph and tree operators.

At this point, another change had to be made, to guarantee that no invalid solutions are produced. In one hand, in graph-based operators, rule names do not really matter, because a correspondence list is built upon crossover application and connections re-established. In the other hand, considering the tree-based approach, CFDG rule names are of huge importance, as later, in mutation, <NONTERMINAL> subtree nodes can be swapped, linking to non-existing rules. So, when using graph-crossover with tree-mutation it is important to make sure that the NONTERMINAL derivation rule is updated, according to the individual's rule names. From this, another problem emerges. If rule names are not the same in both individuals being crossed over, the generation of invalids may occur. To cope with that, a repair procedure was developed, which changes all the calls to non-existing rules by valid ones.

In the following sections the parameterisation of the experiments is detailed (Section 4.1.4.1) and a thorough analysis of the results is carried out (Section 4.1.4.2).

4.1.4.1 Experimental Setup

The setup herein detailed, Table 4.3, merges the configurations from both approaches, when ran in isolation. Additionally, as the initial population is defined using the procedure from the tree-based approach, we decided for the use of the pre-grammar introduced in Grammar 4.1, for the same reasons as in Section 4.1.3.1.

As one may notice, tests will be performed with slightly longer runs (300 generations instead of 100). This decision was taken with regard to the experimental results

Mutation	Crossover	JPEG Size	Combined
Graph	Tree	$32800.133^{(1)(2)}$	$0.01123433^{(1)(2)(3)}$
Graph	Graph	$31769.367^{(3)(4)}$	$0.01006618^{(3)(4)(5)}$
Tree	Tree	$37260.567^{(2)(3)(5)}$	$0.01316316^{(2)(4)}$
Tree	Graph	$35959.167^{(1)(4)(5)}$	$0.01355605^{(1)(5)}$

Table 4.4: Fitness of the best individual for each of the possible operators combinations, using as fitness function the *JPEG Size* and a combination of *Complexity*, *Bell* and *Contrasting Colours*. Results are averages of 30 independent runs. Within each column, a digit is placed next to a pair of setups whose results are statistically different.

of the previous sections where, in some cases, longer runs could have led to better insights over the results.

Taken all this into consideration, in the next section, results of the following experiments will be discussed: tree and graph mutation with tree and graph crossover and a combination of tree mutation with tree and graph crossover, where both crossovers are applied during the experiments. In this last scenario, before applying an operator, it is first needed to decide if mutation, crossover or reproduction is going to be used; if the decision favours crossover it is then necessary to decide whether tree or graph crossover is to be employed, using the probabilities defined in Table 4.3 (last row).

4.1.4.2 Experimental Results

Following what was mentioned in the last section, we carried out experiments using all the possible pairs of mutation and crossover operators, namely: (graph-mutation, tree-crossover), (graph-mutation, graph-crossover), (tree-mutation, tree-crossover), (tree-mutation, graph-crossover). We have also taken into consideration the difficulty of evolving a certain fitness function and, as such, tests were performed using two distinct fitness functions: *JPEG Size* and a combination of *Complexity*, *Bell* and *Contrasting Colours*.

Table 4.4 shows the results of these tests in what concerns the maximum reached fitness (averages of 30 runs). Within each column, a digit (e.g. 1) is placed next to a pair of setups whose results are statistically different. For example, the (1) in the JPEG Size column denotes that the graph-mutation / tree-crossover setup is statistically different from tree-mutation / graph-crossover one.

A brief perusal of the results immediately presents the graph-mutation as worse than the tree one. All experiments where graph-mutation is used provide lower re-

Tree-Crossover	Graph-Crossover	JPEG Size	Combined
0	1	35959.167$^{(1)}$	0.01355605
0.3	0.7	35190.2	0.01321375
0.5	0.5	36954.767	0.01360797
0.7	0.3	37287.23	0.01363978
1	0	37260.567$^{(1)}$	0.01316316

Table 4.5: Fitness of the best individual using tree-mutation and both crossover operators. Two fitness functions were used: *JPEG Size* and a combination of *Complexity*, *Bell* and *Contrasting Colours*. Results are averages of 30 independent runs. Within each column, a digit is placed next to a pair of setups whose results are statistically different.

sults than those where tree-mutation is used; moreover, this difference is statistically significant.

Focusing on *JPEG Size* results, apart from the previously stated conclusion, it is also possible to claim that the best results were obtained with the tree-mutation / tree-crossover setup, which is statistically better than the tree-mutation / graph-crossover. On the other hand, it is not possible to state, with confidence, which combination of operators leads to the best results in the experiments where *Complexity*, *Bell* and *Contrasting Colours* are used as fitness function, because no statistically difference exists between the two best combinations. Although, it is possible to infer that graph-mutation / graph-crossover is worse than graph-mutation / tree-crossover, establishing it as the worst option for guiding evolution with this fitness function.

Bearing the previous conclusions in mind, and with the intention to try to better understand which combination of operators is must suited for multiple fitness functions, we designed tests where we used tree-mutation in conjunction with both graph operators, under different probabilities. Results of these tests are presented in Table 4.5.

Once again, other than the marked pair, there are no statistical differences in the results. However, a clear trend is visible. Increasing the tree-crossover probability (consequently reducing the graph-crossover one) promotes higher fitness values. At this point we decided, for this set of experiments, to focus on the evolution of the population's average fitness across generations. Figures 4.20 and 4.21 depict these results for the *JPEG Size* and the combination of *Complexity*, *Bell* and *Contrasting Colours*, respectively.

The trend that was already visible now becomes highly noticeable. Statistical tests confirm it; although, there are less tests that show statistical difference in the

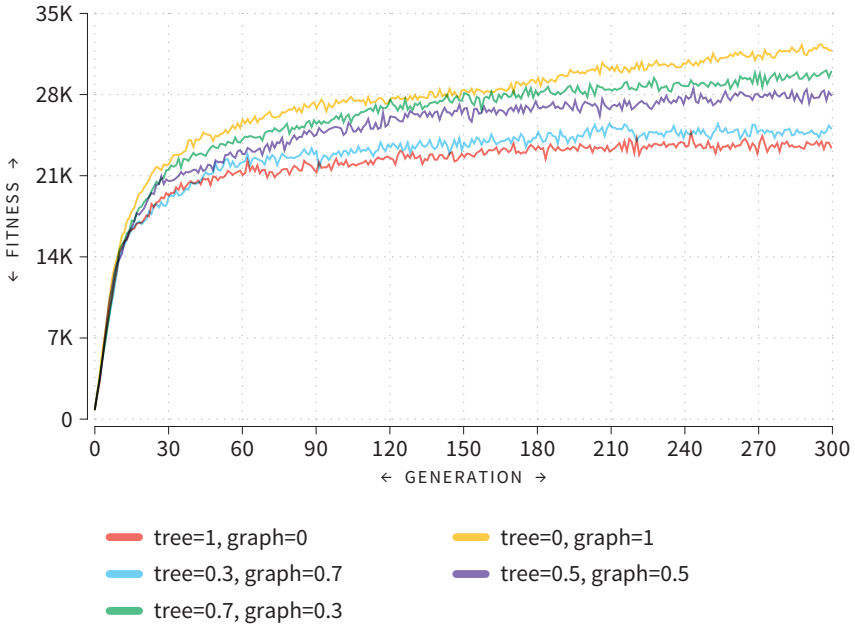


Figure 4.20: Evolution of the average fitness across generations using *JPEG Size* as fitness function. Results are averages of 30 independent runs.

evolutionary runs with the combined fitness function than when just using the *JPEG Size*, which also confirms the difficulty of the problem.

From all this data it is then possible to appoint the tree-mutation / tree-crossover as the best setup.

4.2 Evolving Musical Sequences

In order to compose and evolve musical sequences we first need to choose a way of representing and reproducing them. That is the scenario under which Musical Instrument Digital Interface (MIDI) [1] appears. Among many functionalities, MIDI allows the composition and storage of electronic music, through a wide set of control and data commands.

In simple words, MIDI files store status and data bytes. The first ones define the type of message that is then followed by a variable number of data bytes. These types of messages are what control each one of the possible 16 MIDI channels. The maximum number of 16 channels means that we can play a maximum of 16, from the available 128, different instruments independently, i.e., each channel corresponds to an instrument. Usually, the 10^{th} channel is reserved to percussion instruments.

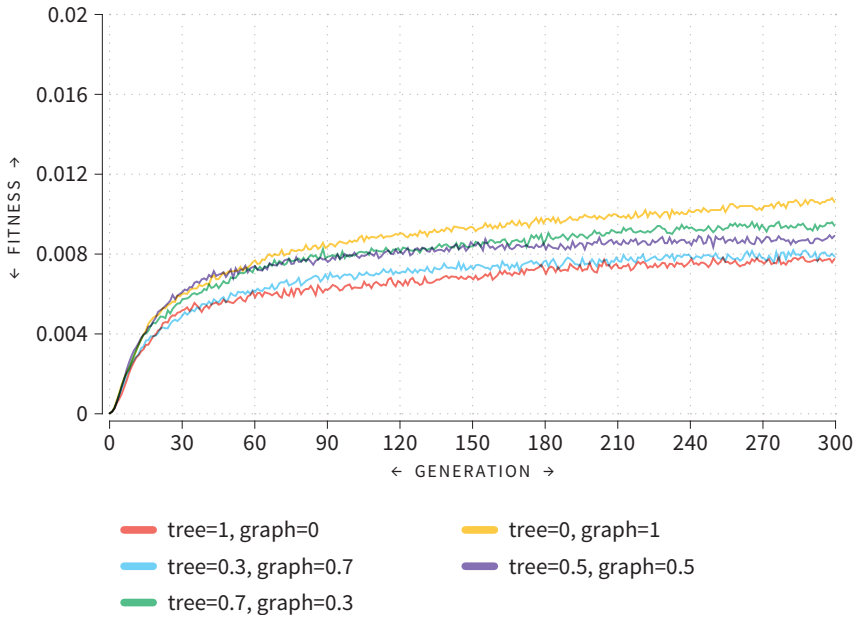


Figure 4.21: Evolution of the average fitness across generations using the combination *Complexity*, *Bell* and *Contrasting Colours* as fitness function. Results are averages of 30 independent runs.

The two most important commands are note on and note off, and they are the ones responsible for playing a sound in a specific channel. To define a note, we need its pitch and velocity. Pitch value defines the frequency of the note to be played, whereas the velocity defines the strength with what it should be played (in most systems this can be translated into volume). The difference in time between the note on and note off commands describes the note duration.

It is important to mention that MIDI files are not playable by themselves. If one wants to listen the sequence of notes stored in this format, a synthesiser must be used. To some extent, this can be considered as a drawback because, different synthesisers can have different sounds for each instrument, turning the quality of reproduction variable, in accordance to the quality of the synthesiser.

There is much more to explain about the MIDI protocol. Although, MIDI is not the true aim of this Dissertation and, as so, considered out of scope. Having explained the concepts necessary for the understanding of the following sections, if more information is sought refer to [21].

With the objective of evolving musical sequences, encoded as MIDI files, we will, in the upcoming sections, show how we have used the previously described approaches

(Chapter 3) to accomplish it. We decided to focus on the method where components from both of them are merged, so that we assess the capacity of both operators to promote evolution, at once.

In Section 4.2.1 the grammar representation of the individuals will be addressed. Section 4.2.2 details the mapping from the grammars to MIDI. Following, the experimental setup (Section 4.2.3) and results (Section 4.2.4).

4.2.1 Grammar Representation

Analogously to CFDGs, we propose an augmented context free grammar capable of representing sequences of notes. This augmented grammar is formed by a 5-tuple (V, F, Σ, R, S), where V, Σ, R and S have the same meaning as in CFDGs. The new symbol, F, stands for a set of functions, that grammatically act similarly to non-terminals, namely:

- mirror(arg) – reverses the order of arg;
- playandmirror(arg) – plays arg and then its reverse;
- playtwice(arg) – arg is played two times in a row;
- playtwo(arg, arg) – the two input arguments are played in simultaneous;

where $arg \in (V \cup F)$. This means that the arguments received by the functions can be any non-terminal or function. When functions are nested, for example: playtwice(mirror(non-terminal)), the inner functions have to be evaluated first. For that, in the example, before applying the playtwice function it is needed to assess the output of mirror(non-terminal). The rationale of functions relates to the need of giving some sort of structure to what the user will ultimately listen.

In this particular domain only the terminal symbols note and chord are needed ($\Sigma = \{note, chord\}$). Moreover, assuming that the volume is always 100%, to define a note, at least two values are needed, its pitch and duration. Grammatically, this requirement can be seen as two mandatory parameters associated with the calls to terminals because, as previously mentioned, the grammar is augmented. To define a chord, analogously to note, the base pitch and duration of the chord must be specified, as two mandatory parameters. The chord is later mapped into three notes played at the same time, with the same duration and a defined difference of pitch between them.

Having that said, we are only missing the non-terminals (V). In CFDGs non-terminals perform calls to multiple non-terminals and terminals. Here, the rationale

```
start A

rule A{
    note {pitch 60 duration 1}
    B {}
    playtwice (B) {pitch 1 duration -0.125}
}

rule B{
   note {pitch 65 duration 0.5}
   chord {pitch 65 duration 1}
   mirror (B) {}
}
```

Figure 4.22: Example of a grammar capable of representing a musical sequence.

is exactly the same, with the addition of calls to functions. This way, whereas in CFDGs non-terminals would stand for a part of the image, here non-terminals encode a subsequence of notes, which is part of the main musical piece. The start symbol (S) is randomly chosen from the set of non-terminal symbols.

Likewise to CFDGs, it is also possible to define the same non-terminal more than once, providing in this way the grammar of a non-deterministic nature.

Both non-terminal and functions calls can be also parameterised. For simplicity reasons we only allow pitch and duration as parameters. When used, they can increase or decrease the pitch and duration of the terminals note of the calling target.

Figure 4.22 shows an example of a grammar that complies with the previously enunciated principles. The way to interpret this grammar is explained in Section 4.2.2.

4.2.2 From Grammars to MIDIs

The method for converting from a grammar to a sequence of notes is in all similar to what happens with Context Free Art.

Beginning from the start symbol, an ordered sequence of notes, chords (terminal symbols) and non-terminal symbols is created. It is considered that, after processing a function, its output is a set of non-terminals that is added to the ordered sequence. This sequence is then traversed multiple times, until no non-terminals are left to expand, or a maximum number of expansions is reached ($maxexpansions$). Setting a maximum number of expansions works as a way of dealing with loops.

The procedure of expansion of a non-terminal must take into consideration the non-deterministic nature of the grammar, i.e., all rules with the same name as the non-terminal participate into a roulette wheel selection, where its probability of being

chosen is the one defined after the rule name (see Figure 4.22). If no value is specified, a default of 1 is assumed.

Algorithm 6 Mapping of an individual to a sequence of notes.

```
procedure NotesSequence(ind, maxexpansions)
    sequence ← ExpandNonTerminal(ind.rules, ind.startshape)
    expansions ← 1
    while sequence has non-terminals and expansions < maxexpansions do
        for symbol in sequence do
            if symbol is non-terminal then
                subsequence ← ExpandNonTerminal(ind.rules, symbol)
                Replace(symbol, subsequence)
                expansions = expansions + 1
                if expansions > maxexpansions then
                    break
                end if
            end if
        end for
    end while
    return sequence
end procedure
```

It is also needed to keep in mind the parameters used in functions and calls to non-terminal symbols. These, as previously mentioned, affect the pitch and duration of the notes of the target non-terminal symbol. For example, in the call that is made to non-terminal B, from rule A (see Figure 4.22), the actual pitch and duration of the first note are 65 and 0.5, respectively. Although, when calling the non-terminal B that is passed as argument to the function playtwice, the pitch and duration of the first note of non-terminal B are 66 and 0.375, respectively. In another hypothetical situation, if we have a rule A, calling a rule B, which by consequence calls a rule C, the notes in rule C must consider the parameters passed from A to B and from B to C, i.e., parameters used in calls are cumulative.

The above mentioned principles are presented, in a more formal way, in Algorithms 6 and 7. The first one details the complete procedure of mapping from an individual's genotype into the sequence of notes. The other focuses the expansion of a non-terminal symbol.

At this point, we have a set of notes (output from Algorithm 6), which can be easily saved to MIDI. To accomplish that, we used a Python's library[3]. Adopting

[3]MIDIUtil – https://code.google.com/p/midiutil/

Algorithm 7 Expansion of a non-terminal symbol.

```
procedure ExpandNonTerminal(productionrules, rulename)
    sequence ← EmptyList()
    rule ← ChooseExpansion(productionrules, rulename)
    for call ∈ rule do
        if call is terminal or call is non-terminal then
            sequence.add(call)
        else
            sequence.extend(ExpandFunction(call))
        end if
    end for
    return sequence
end procedure
```

Evolutionary Engine	Values
Number of runs	10
Number of generations	15
Population size	20
Graph-crossover probability	0.18 (0.6 × 0.3)
Tree-crossover probability	0.42 (0.6 × 0.7)
Tree-mutation probability	0.3
Tournament size	2
Elite size	Top 5% of the population
Grammar Parameters	**Values**
Maximum number expansion steps	10
Limits of the calls transformations	pitch ∈ [-2,2], duration ∈ [-0.25,0.25]
Terminal symbols	note
Functions	playtwice, mirror, playandmirror, playtwo

Table 4.6: Parameters used for the experiments evolving musical sequences.

this library, to produce a MIDI file we only need to provide, for each note, its pitch, starting time, duration, channel and volume. For simplicity reasons, we assume that the volume is always 100% and that we are only playing an instrument (piano), so we only need one channel.

4.2.3 Experimental Setup

As previously mentioned, we decided to focus on the method where components of both approaches from Chapter 3 are merged. We chose not to carry full extensive tests with all possibilities (as done with CFDGs) because the aim of this section is different. We only want to prove that it is possible to apply the evolutionary engine to different domains and, as so, the experiments herein presented work as a proof of concept.

```
<S> ::= <STARTSHAPE> <RULE>(3, NONTERMINAL, STRING, True)
        <RULE>(1,3)
<STARTSHAPE> ::= startshape <NONTERMINAL>
<RULE> ::= rule <NONTERMINAL> <PROB> { <CALLS> }
<PROB> ::= [0,1] | λ
<CALLS> ::= <SYMBOL> <CALLS> | <SYMBOL> <CALLS>
         |  <SYMBOL> <CALLS> | λ
<SYMBOL> ::= <TERMINAL> | <TERMINAL> | <TERMINAL>
          |  <NONTERM> | <NONTERM> | <FUNCTION>
<TERMINAL> ::= note { pitch <PITCH> ; duration <DURATION> }
            |  note { pitch <PITCH> ; duration <DURATION> }
            |  note { pitch <PITCH> ; duration <DURATION> }
            |  chord { base <PITCH> ; duration <DURATION> }
<PITCH> ::= 60 | 61 | 62 | 63 | 64 | 65
         |  66 | 67 | 68 | 69 | 70 | 71
<DURATION> ::=  1.0 | 0.5 | 1.0 | 0.5
            |   1.0 | 0.5 | 0.25 | 0.125
<NONTERM> ::= <NONTERMINAL>  <PARAMETERS>
<FUNCTION> ::= playtwo ( <FUNCARG> ; <FUNCARG> )  <PARAMETERS>
            |  playtwice ( <FUNCARG> )  <PARAMETERS>
            | mirror ( <FUNCARG> )  <PARAMETERS>
            | playandmirror ( <FUNCARG> )  <PARAMETERS>
            | playtwice ( <FUNCARG> )  <PARAMETERS>
            | mirror ( <FUNCARG> )  <PARAMETERS>
            | playandmirror ( <FUNCARG> )  <PARAMETERS>
            | mirror ( <FUNCARG> )  <PARAMETERS>
            | playandmirror ( <FUNCARG> )  <PARAMETERS>
<FUNCARG> ::= <FUNCTION> | <NONTERMINAL> | <NONTERMINAL>
<PARAMETERS> ::= pitch [-2, 2] <PARAMETERS>
              |  duration [-0.25, 0.25] <PARAMETERS>
              |  λ
```

Grammar 4.3: Pre-grammar used for the evolution of grammars capable of representing musical sequences.

Being a proof of concept we opted for guiding evolution in a user-guided manner. One may argue that, by doing so, results are subjective to the users aesthetic preferences. That is correct but, once again, this is a proof of concept. The aim is only to assess the adequacy of the system to evolve candidate solutions in other domains, regardless of the used fitness function.

In the used evolutionary algorithm, individuals are represented by derivation trees of an a-priori defined pre-grammar and are mapped into a graph representation before

application of the graph-crossover. For that, the first step is the definition of a pre-grammar capable of providing grammars such as the one presented in Figure 4.22. The used pre-grammar is presented in Grammar 4.3. As seen, it contains a lot of repeated productions. This was done with purpose, in a way to bias the grammar, in order to produce more audible note sequences, avoiding too much repetition or notes played at once. More about the motivation for the grammar biasing will be discussed in the next section.

Table 4.6 details the remaining parameters needed for the performance of experiments. The low value in the number of runs, generations and population size are a result of the user-guided fitness. Just for this setup, the user is required to ear about 3000 MIDI files, classifying each one of them, in a scale from 0 to 5, with regard to all the remaining. This can easily be understood as an exhausting process.

4.2.4 Experimental Results

In this section we will discuss and present the results obtained while testing the evolution of musical sequences. To start, the motivation that let to biasing the grammar and how it was accomplished will be described. Later, we will show samples of the evolved musical pieces, using a user guided-method that, despite subjective, we considerer convincing to prove the capacity of the evolutionary engine to cope with different domains.

An unbiased pre-grammar of the one shown in Grammar 4.3 can be understood as composed by the exact same production rules, although without repetitions in its derivation steps. For example, the production rule <RULES> would consist of the derivations <SYMBOL> <CALLS> and λ, with no repetitions of <SYMBOL> <CALLS>. The same reasoning should be applied for the remaining production rules. From that, an important consequence emerges; when deciding which derivation step should replace the non-terminal, they all have the same probability.

As a result, using the unbiased grammar, the following happens. When expanding a rule, i.e., adding calls to it, the same chance is given to λ and <SYMBOL> <CALLS>. In theory this means 50% of the rules will be empty (no calls to terminals, non-terminals or functions). Furthermore, when expanding a function, its argument can be a non-terminal or another function. Being the probabilities the same, it leads to multiple nested functions, which hardly produce interesting musical compositions. The first pointed situation guides to the output of sequences of notes that are either too simple, or even empty, due to the lack of calls to terminals, non-terminals and functions. In the other hand, it easily produces highly complex musical pieces, such

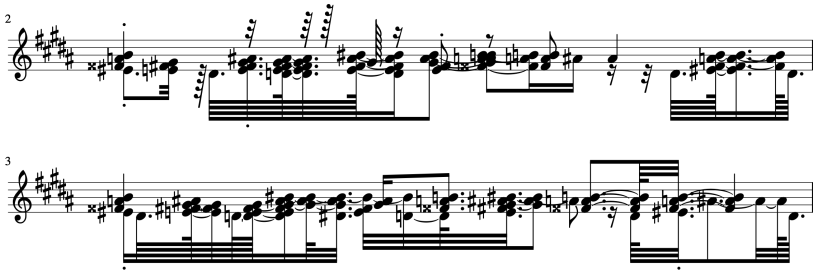


Figure 4.23: Example of part of the sequence of notes produced by an individual generated using the unbiased version of Grammar 4.3.

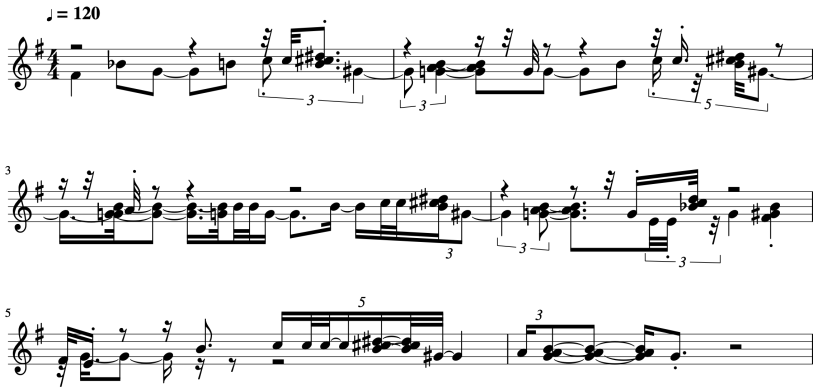
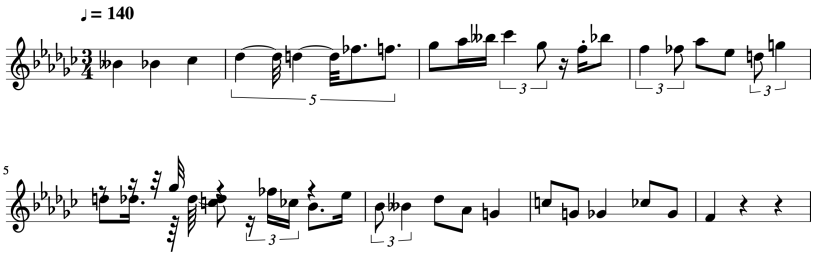


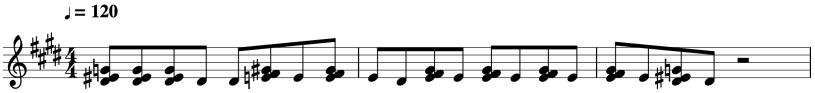
Figure 4.24: Example of a sequence of notes produced by an individual considered of high-quality, generated using the unbiased version of Grammar 4.3.

as the one depicted in Figure 4.23, where constantly there is a high number of notes played at the same time. This is often the consequence of nested playtwo function calls. In the example it is also clear the dominance of short duration notes, because in the unbiased grammar we were allowing shorter time durations (0.0625 and 0.03125), which were later discarded. Despite all that, using an unbiased grammar it is also possible to evolve simple aesthetic musical pieces, such as the one in Figure 4.24.

Biasing the grammar arises as a solution for the above mentioned problems. Although, it is intrinsically related with the user's aesthetics notion. Following this line of thought, we increased the probability of creating calls to non-terminals, terminals and functions, by making <SYMBOL> <CALLS> present in the grammar more than once. We also enhanced the type of call that should be made, giving higher rates to terminals, non-terminals and functions, in this order. Focusing terminals, we established that the terminal note should be used more often than chord, in order to discourage the noise produced by a huge amount of notes played at once. Furthermore, with the same problem in mind, the function playtwo is the one with the lowest probability of being used. Playtwice, also has lower probability of being picked



(a) Example A



(b) Example B

Figure 4.25: Example of a sequence of notes produced by an individual considered of high-quality generated using Grammar 4.3.

than the others, although higher than playtwo, because it leads to the same pattern over time, producing monotonous sequences of notes. Function arguments were biased, setting a preference for non-terminals, avoiding this way wide nested functions. Action was also taken to change the notes duration, after noticing that low values are not perceptible when playing the MIDI file. For that, the options focus higher durations.

After introducing the above mentioned changes into the used pre-grammar, more of the output results were considered useful. To start, there were less empty and highly complex note sequences. It was possible to actually see that, across generations an improvement in the quality of the individuals was being accomplished, leading to the evolution of simple and structured musical pieces. Examples of outputs of the system are shown in Figure 4.25. In the attached CD, we include an evolutionary run performed using the setup specified in Section 4.2.3 as well as more samples of individuals that were considered of high quality by the users, during the evolutionary process.

4.3 Conclusions

In the present chapter we perform multiple experiments over the approaches detailed in Chapter 3. This tests focus two different domains.

First, we try the evolution of Context Free Design Grammars that, when rendered, give origin to images in the PNG format. With this domain, both graph and tree-based approaches were tested. It was proved that one and the other are capable of evolving CFDGs, with all the used fitness functions except two (*Fractal Dimension*

and *Lacunarity*) which were later ruled out, for being inappropriate for the assessing of fitness of non-fractal images. As so, they are ill-suited to promote the evolution of non-fractals into fractal images. We also demonstrated the capacity of the system to guide evolution according to a fitness which takes several others into account producing results that must bespeak characteristics of each one of them.

After showing that, in isolation, both approaches work as expected, we tried to merge operators from each one of them, i.e., try to evolve CFDGs for example with tree-mutation and graph-crossover. The procedure for population initialisation was fixed to the tree-based one, due to generalisation purposes. For that, individuals are derivation trees and, as so, before applying any graph operation a mapping from a tree to a graph is needed, converting then them back to a tree. Results clearly show that when using graph-mutation results are worse. They also establish the tree-crossover as the best crossover operator. For that, in the next chapter, solutions to improve the performance of the graph-crossover are going to be explored.

To end, experiments in another domain, namely music, were carried out. The goal of this last section of the chapter is to assess the capacity of the evolutionary engine to deal with different environments. For that, it can be perceived as a proof of concept and, therefore, only one of the above discussed approaches was used; more precisely, the one that combines tree-mutation with tree and graph crossovers (preference set to tree-crossover). After introducing bias in the used grammar, showing that in some scenarios it can be useful to lead to aesthetic results, we performed user guided runs. Even tough the results can be seen, up to some extent, as subjective, we demonstrate that the method is adequate as, over time, the output musical sequences are considered of higher quality.

Chapter 5

Alignment

Alignment can be defined as the procedure that establishes a mapping between two structures, such as trees or graphs. Moreover, the output mapping should aim at maximising the similarity among the parts of each structure that are to be aligned. In other words, it should try to preserve most of the characteristics of the structures, i.e., blocks in one structure should be mapped to similar ones in the other.

The most common way to measure similarity is using topological metrics. All methods within this class of approach see nodes as equal parts and compute their similarity with regard to the relation between them.

On the other hand, we have procedures which take into consideration only the similarity among nodes. Although, in some situations, this type of methods fail for being domain dependent, requiring a-priori knowledge.

The optimal way to tackle this problem is, in our opinion, to merge the concepts from the previous paragraphs, i.e., consider both topological and node similarities.

The following sections are organised as follows. Section 5.1 focuses methods for graph alignment. Algorithms for the alignment of trees are not going to be explored, because derivation trees are constrained to ensure the generation of valid individuals, by restricting cutting points to nodes with the same derivation rule. From the previous, the application of any sort of alignment is not expected to generate any better results. The objective of this study is then to try improving the graph-crossover performance, by choosing cutting points in a more informed way. By doing so, we can pick the first crossover point at random and the second one as its alignment or we can go for the alignment that maximises the similarity measure. These possibilities will be addressed in the experimentation section, Section 5.2. To end, conclusions will be drawn (Section 5.3).

5.1 Graph Alignment

Multiple algorithms for aligning graphs exist, most of them divided into two phases. Usually, the first one focuses in computing the cost of aligning every pair of nodes of the two graphs, while the second extracts the actual alignment from the previously built cost matrix.

Regarding the alignment method, it should be able to produce a one-to-one mapping between the two graphs that are being aligned. Most often than not, that is impossible because the number of nodes in each structure is different. In that case, a one-to-one mapping from the smallest to the largest structure is sought, i.e., every single node of the graph with less nodes should be aligned to exactly one node of the graph with more nodes.

Taking all the above into account, and after analysing the survey by Döpmann [16], where several graph alignment algorithms are benchmarked, we opt by adapting the Graph Aligner (GRAAL) algorithm [33] to our current scenario.

GRAAL, as the majority of the alignment algorithms follows a two phase structure where, to produce the cost matrix, both topological and node similarities are considered.

Denoting by G and H the two graphs we aim at aligning, the topological similarity (top_{sim}) between each pair of nodes $v \in G$ and $u \in H$ is computed according to the following equation:

$$top_{sim}(v,u) = 1 - \frac{deg(v) + deg(u)}{max\_deg(G) + max\_deg(H)}, \tag{5.1}$$

where deg represents the degree of a node in the graph and max_deg the maximum degree of a graph. Consequently, the output is a value in the interval $[0,1]$, where lower values stand for nodes that are more similar.

At this point, we introduce changes in the algorithm. In the original paper, node similarity is computed using the signature similarity algorithm [48], that measures local topological similarity between two given nodes, taking into account their graphlet degrees (for more informations refer to [48]). In order to align the approach with our current scenario, to assign similarity to two nodes, we ask the user to initially define a dissimilarity matrix, where every possible terminal, non-terminal and parameter that the evolutionary engine is allowed to use are compared. This way, every cell in the matrix means how dissimilar two symbols are. The higher the value in the cell is, the more dissimilar the elements are. For the fact that one node can have more

terminals and non-terminals than the other, the comparison to λ is also contained in the dissimilarity matrix. The same rationale is applied to parameters.

Node similarity is then calculated by comparing the terminals, non-terminals and parameters from each pair of nodes, scoring their alignment with regard to the dissimilarity matrix defined by the user. Efforts to scale the output value to the $[0, 1]$ interval were also made. As in the topological similarity, values close to 0 represent nodes that are closer to each other.

If we then denote by $node_{sim}$ the node similarity we can compute the cost of aligning each node v of the smallest graph to a single node u of the largest one as:

$$cost(v, u) = \alpha \times top_{sim}(v, u) + (1 - \alpha) \times node_{sim}(v, u), \tag{5.2}$$

where $\alpha \in [0, 1]$ represents the weight given to the topological part of the alignment. Moreover, as both top_{sim} and $node_{sim}$ output values are between 0 and 1 it is easy to understand that the output of the *cost* function will also be within the same interval.

Algorithm 8 Graph alignment.

```
procedure GraphAlign(G, H, CostMatrix)
    alignments ← EmptyList()
    marked_G ← EmptyList()
    marked_H ← EmptyList()
    while |marked_a| < |G| do
        (node_G, node_H) ← FindLowestCost(marked_G, marked_H, CostMatrix)
        marked_G.append(node_G)
        marked_H.append(node_H)
        alignments.append((node_G, node_H))
    end while
    return alignments
end procedure
```

With the end of the first phase, which terminates with the calculation of a cost matrix for all pairs of nodes, it is now possible to move to the actual alignment phase, where a mapping based on the previously computed heuristics will be performed. Algorithm 8 depicts how the one-to-one mapping is computed. In simple words, the lowest cost alignment will be chosen until all nodes from the smallest graph (G) are aligned to exactly one node of the largest graph (H).

The produced alignment can then be used to pick the crossover point in a more informed way. In one hand we can go for the alignment that has the lowest cost, i.e., the first element of the list returned by Algorithm 8. In the other hand, we can select

from the smallest graph one node randomly, and then choose as crossover point, in the other graph, the one to which it is aligned to. Experiments focusing alignment where one pair of nodes is chosen at random will be performed in the next section. This pair of nodes will then be used as the cutting point for the graph-crossover operator. We avoid choosing the best possible pair of nodes because that will lead to a deterministic process, which in evolutionary algorithms can easily conduct to stagnation of the evolution. We will also address different options for computing the degree of a node (used in topological similarity). More precisely, we will test the computation of the degree considering only the incoming connections of the node, only the outgoing connections and both.

5.2 Experimentation

As mentioned in the previous section, the followed alignment methodology consists of two important components: (i) topological similarity, which assesses the affinity of two nodes with respect to the existing relations between them; (ii) node similarity, that takes into account the domain of the problem and measures the likeliness of two nodes, concerning their meaning, i.e., what they represent in the environment context.

First, we performed tests considering just the multiple alignment possibilities, i.e., instead of incorporating alignment in the evolutionary algorithm we decided first to run it in isolation. That is, we create a population composed of randomly generated individuals and then, we apply to all of them the graph-crossover operator, choosing the cutting points using different versions of alignment. Then, for each setup we assess its constructive rate (percentage of crossovers that generate offspring greater or equal, in terms of fitness, than the worst of the parents) and average fitness. Each one of the tests is then compared with the same setup under the standard graph-crossover, i.e., the one without any sort of alignment.

To better understand the impact of alignment, when testing its different possibilities (topological and node similarities) we varied several parameters. The first one that we tested is the number of nodes in a graph. It is expected that alignment performs better when applying crossover to large graphs than the opposite because, if the number of nodes is low, it is likely that choosing the cutting points at random leads approximately to the same results than when choosing them in a wiser manner. Finally, we tested two forms of sorting the set of individuals to whom crossover will be applied. The two possibilities under study are no sorting at all (i.e., they are

Evolutionary Engine	Values
Number of runs	30
Number of generations	300
Population size	100
Graph-crossover probability	0.6
Tree-mutation probability	0.3
Leaves probability	0.3
Tournament size	5
Elite size	Top 2% of the population
CFDG Parameters	Values
Maximum number expansion steps	100000
Limits of the geometric transformations	Check used pre-grammar
Limits of the colour transformations	Check used pre-grammar
Terminal symbols	Check used pre-grammar
Alignment Parameters	Values
Number of nodes	{10, 30, 100}
Node degree	incoming connections, outgoing connections, incoming and outgoing connections
Terminals and non-terminals dissimilarity matrix	Check Matrix 5.1
Parameters dissimilarity matrix	Check Matrix 5.2
α	0.5

Table 5.1: Parameters used for the graph alignment experiments.

essentially in a random order) or sorting them by fitness. The second choice implies that individuals that have higher quality values are always crossed with highly fitted individuals too, and the opposite. If no sorting is used, individuals of different quality are breed.

Results of these experiments are presented in Sections 5.2.2, 5.2.3 and 5.2.4, respectively with, topological similarity, node similarity and topological and node similarities. Then, in Section 5.2.5 we will integrate the best of the alignment alternatives into the evolutionary algorithm previously tested in Chapter 4 (tree-mutation / graph-crossover).

5.2.1 Experimental Setup

The objective of this chapter is to try to increase the performance of the evolutionary engine by choosing the graph-crossover cutting points more wisely. From that, we opt first to try to figure out which of the alignment possibilities produces the best results, incorporating it later in the evolutionary engine (tree-mutation and graph-crossover) for the evolution of CFDGs.

Table 5.1 details the needed parameterisation for the evolutionary engine and to the alignment methods. To initialise the population we use the same pre-grammar as in previous experiments (Grammar 4.1).

As mentioned in the previous section, for computing the similarity between nodes a dissimilarity matrix must be defined. Matrices 5.1 and 5.2 compare all possible calls and parameters, respectively. The rationale behind the first one is that the difference between terminals and non-terminals should be higher than the one between two terminals or two non-terminals. Moreover, it is considered that it is better to have a terminal aligned with a non-terminal than with nothing (λ). To propose Matrix 5.2 we divided the parameters into two main groups: geometric (geom = transl $\cup$ transf) and colour (col = {brightness, hue, saturation, alpha}). Furthermore, it is possible to subdivide the geometric ones in: translations (transl = {x, y, z}) and transformations (transf = {skew, flip, rotate, size}). Then, denoting by C the cost of alignment, we established that the following conditions have to be met:

$$C(geom, col) > C(geom, geom) = C(col, col),$$
$$C(transf, transl) > C(transl, transl) = C(transf, transf).$$

	non-terminal	SQUARE	TRIANGLE	CIRCLE	λ
non-terminal	0	2	2	2	3
SQUARE	2	0	1	1	3
TRIANGLE	2	1	0	1	3
CIRCLE	2	1	1	0	3
λ	3	3	3	3	0

Matrix 5.1: Terminals and non-terminals dissimilarity matrix.

	x	y	z	skew	flip	rotate	size	brightness	hue	saturation	alpha	λ
x	0	2	2	2.5	2.5	2.5	2.5	4	4	4	4	3
y	2	0	2	2.5	2.5	2.5	2.5	4	4	4	4	3
z	2	2	0	2.5	2.5	2.5	2.5	4	4	4	4	3
skew	2	2	0	2.5	2.5	2.5	2.5	4	4	4	4	3
flip	2.5	2.5	2.5	0	2	2	2	4	4	4	4	3
rotate	2.5	2.5	2.5	2	0	2	2	4	4	4	4	3
size	2.5	2.5	2.5	2	2	0	2	4	4	4	4	3
brightness	4	4	4	4	4	4	4	0	2	2	2	3
hue	4	4	4	4	4	4	4	2	0	2	2	3
saturation	4	4	4	4	4	4	4	2	2	0	2	3
alpha	4	4	4	4	4	4	4	2	2	2	0	3
λ	3	3	3	3	3	3	3	3	3	3	3	0

Matrix 5.2: Parameters dissimilarity matrix.

5.2.2 Topological Similarity

Like mentioned earlier, when performing alignment with topological similarity, part of its computation relies on the degree of the nodes being aligned. As we are applying it to directed graphs, several possibilities for assessing the degree of a node exist, namely:

Incoming Connections – the degree of a node corresponds to the number of connections that reach it. For example, the node *Pelo* of Figure 4.3 has 5 incoming connections and, as so, a degree of 5;

Outgoing Connections – the degree of a node is equal to the number of links that start from it. In Figure 4.3, *Pelo* has an outgoing degree of 1 (note that the calls to terminals are not depicted in the figure);

Incoming and Outgoing Connections – the degree of the node corresponds to the number of incoming and outgoing connections. Using this option, node *Pelo* of Figure 4.3 has a degree of 6.

In order to try to figure out which of the options for computing the node degree is best, we performed a wide range of tests using all of them. Results are shown in Table 5.2. The ones that are statistically different from the standard crossover (not using alignment), for each set of tests, are marked with an asterisk.

An analysis of the results confirms the expected. Increasing the size of the graph, i.e., the number of nodes, leads to crossover with topological alignment performing better than the one where no alignment is used. Furthermore, it is possible to conclude that the one where the degree of nodes is computed taking into account only the number of outgoing connections outperforms the remaining options, both in terms of constructive rate and fitness. That conclusion arises from the fact that with both population sorting options and different number of nodes, the outgoing connections option is the one that yields, most often, results that are statistically different from the version of crossover without any sort of alignment.

Another interesting point pertains the population sorting. Sorting the population by fitness always provides lower results than without any sorting. This conclusion can be regarded as follows; when combining two individuals with similar fitnesses it is harder to surpass the worst parent's quality. On the other hand, if breeding two individuals with different fitness values, the offspring tends to be better than the worst of the parents. However, this result is expected because it is often harder to overcome two good solutions than one good and one bad.

Num. Nodes	Pop. Sort.	Alignment	Const. Rate (%)	Fitness ($\times 10^{-5}$)
10	Yes	None	70.47	2.94
		Incoming	69.73	3.45
		Outgoing	72.73*	3.55*
		Inc. + Out.	71.47	3.57
	No	None	85.07	3.73
		Incoming	85.23	3.66
		Outgoing	85.13	3.49
		Inc. + Out.	85.43	3.35
30	Yes	None	69.57	3.04
		Incoming	69.7	2.81
		Outgoing	70.3	3.15
		Inc. + Out.	69.47	3.20
	No	None	84.1	3.00
		Incoming	84.43	3.62*
		Outgoing	85.33	3.98*
		Inc. + Out.	84.27	3.85*
100	Yes	None	66.57	2.39
		Incoming	70.33*	3.37*
		Outgoing	69.77*	3.70*
		Inc. + Out.	67.17	2.71
	No	None	83.2	3.07
		Incoming	83	4.01*
		Outgoing	84.57*	3.87*
		Inc. + Out.	81.9	3.18

Table 5.2: Topological similarity alignment results. Cells marked with an asterisk mean that their value is statistically significant when comparing with the crossover without alignment for that set of tests. Results are averages of 30 independent sets of individuals.

From the previous analysis, later, in Section 5.2.4, we will use, in the topological part of the cost function, the option that considers only the outgoing connections of a node to compute its degree.

5.2.3 Node Similarity

Similarly to the previous section, we have also designed experiments focusing only the node similarity of the cost function. As one may remember, before applying this procedure, dissimilarity matrices comparing all possible symbols (terminals and non-terminals) and parameters must be provided to the system. These matrices are presented, respectively, in Matrices 5.1 and 5.2.

Num. Nodes	Pop. Sorting	Alignment	Const. Rate (%)	Fitness ($\times 10^{-5}$)
10	Yes	None	70.47	2.94
		Node Sim.	72.5*	3.17
	No	None	85.07	3.73
		Node Sim.	87.47*	3.50
30	Yes	None	69.57	3.04
		Node Sim.	72.33*	3.08
	No	None	84.1	3.00
		Node Sim.	87.17*	2.85
100	Yes	None	66.57	2.39
		Node Sim.	71.8*	2.99
	No	None	83.2	3.07
		Node Sim.	85.47*	3.23

Table 5.3: Node similarity alignment results. Cells marked with an asterisk mean that their value is statistically significant when comparing with the crossover without alignment, for that set of tests. Results are averages of 30 independent populations of individuals.

Table 5.3 details the results of the performed experiments. From its analysis it becomes clear that node similarity plays an important role in the computation of the cost function. Concerning the constructive rate, in all tested scenarios this form of alignment provides statistically better results than the crossover where no alignment is used. Values related to fitness average do not depict any statistical difference in any of the tests.

Analogously, a relation between the number of nodes and the constructive rate seems to exist. As we increase the number of nodes, constructive rate decreases. In contrast, fitness appears unrelated.

5.2.4 Topological and Node Similarities

Taking into account the experiments performed in the last two sections, where the cost matrix of the alignment is computed only considering one of its components, we now run tests using both. To assess the value corresponding to the topological part we only count the outgoing connections of the nodes, for the reasons above mentioned, in Section 5.2.2.

Table 5.4 details the results of the executed tests. As expected, a mixture between the results of both topological and node similarities, when run in isolation, is obtained. In one hand, the constructive rate of the setups with alignment constantly outperforms the standard crossover. Additionally, with alignment, the average of the

Num. Nodes	Pop. Sorting	Alignment	Const. Rate (%)	Fitness ($\times 10^{-5}$)
10	Yes	None	70.47	2.94
		Node + Top.	73.83*	3.65*
	No	None	85.07	3.73
		Node + Top.	87.00	3.51
30	Yes	None	69.57	3.04
		Node + Top.	72.13*	3.25
	No	None	84.1	3.00
		Node + Top.	85.9	3.75*
100	Yes	None	66.57	2.39
		Node + Top.	71.3*	3.28*
	No	None	83.2	3.07
		Node + Top.	83.57	3.48

Table 5.4: Topological and node similarities alignment results. Cells marked with an asterisk mean that their value is statistically significant when comparing with the crossover without alignment, for that set of tests. Results are averages of 30 independent sets of individuals.

fitness values does usually surpass the tests without alignment too. Furthermore, most of these results are backed up by statistical significant tests, as marked in Table 5.4 cells.

From all the performed tests we then propose the setup where both topological and node similarities are used to compute the cost matrix of the alignment method as the best possible option, for the motives presented in the last three sections. We will now, in the next section, conduct experiments using this type of crossover in the evolutionary algorithm (tree-mutation / graph-crossover).

5.2.5 Alignment Integration

As stated in the previous sections, after successfully employing graph-crossover with alignment over a set of randomly generated individuals we now incorporate it in the evolutionary algorithm.

The cost matrix of the alignment algorithm takes into account topological and node similarities. Moreover, topological similarity is computed just using the outgoing node connections. Graph size was set to 100 because, from previous sections results, it seems to be the most difficult setup to tackle and, as such, the one where the alignment results should be more clearly perceptible. To guide evolution we used a combination of *Bell*, *Complexity* and *Contrasting Colours*.

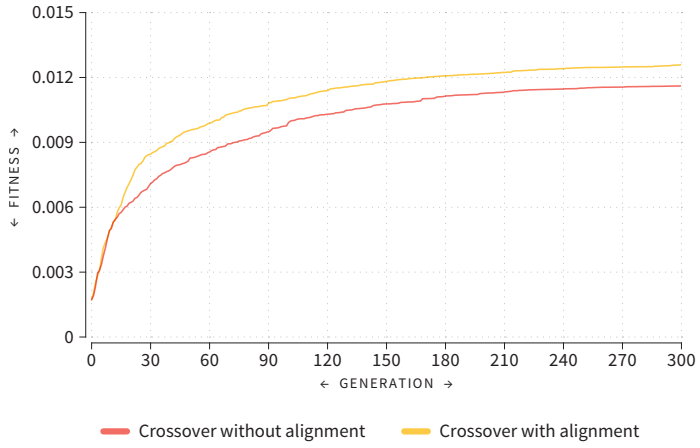


Figure 5.1: Evolution of the fitness of best individual throughout generations with and without alignment employed in the crossover operator. Results are averages of 30 runs.

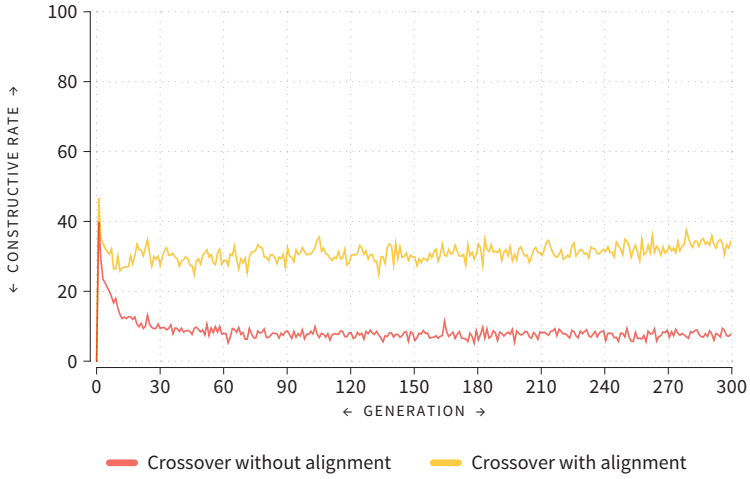


Figure 5.2: Crossover constructive rate throughout generations with and without using alignment. Results are averages of 30 runs.

Results of the evolution of fitness of the best individuals (Figure 5.1) show that, the use of alignment in the crossover operator allows the evolutionary engine to achieve better results. Moreover, the difference between both crossovers, with and without alignment, is statistically significant.

With the intention of understanding what is exactly promoting this difference, we moved into assessing the average percentage of constructive crossover across generations. Results are depicted in Figure 5.2. A perusal analysis clearly shows that, the percentage of individuals generated by crossover that have a fitness greater or equal

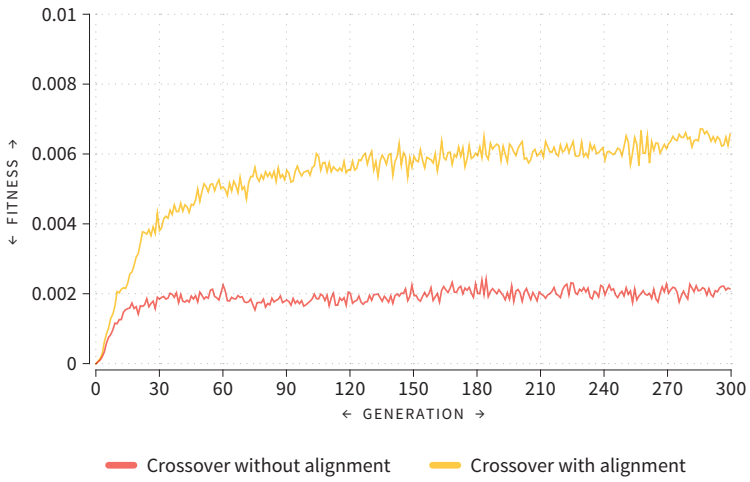


Figure 5.3: Evolution of the fitness of the individuals that are generated by crossover throughout generations with and without using alignment. Results are averages of 30 runs.

than the worst of their parents is far greater using alignment (almost three times superior). Furthermore, they tend to have higher fitness values (Figure 5.3).

From all the previous analysis it is then clear that using alignment improves the quality of the graph-crossover.

5.3 Conclusions

Motivated by the results produced by the graph-crossover we investigate possible approaches for its improvement. One major drawback that seemed clear while analysing the theory behind it is related with the way how its cuttings points are chosen. While in tree-crossover they are picked in a way that assures that at least, the swapped subtrees have some degree of similarity, in graph-crossover that does not happen.

Alignment emerges as a solution for the above problem. By establishing a one-to-one mapping, from the smallest to the largest graph, it returns a list of pairs of nodes that have low alignment cost. Then, from this list, we randomly choose one of the pairs, using its nodes as cutting points.

To compute the cost matrix that is used by the alignment algorithm we adapt the formulation introduced in GRAAL. As such, we consider that, to compute the cost of aligning two nodes, two components are needed: topological and node similarities. While the first considers the connections between nodes, through its degree, the second analyses them in a much fine and semantic level of granularity.

To compute the topological similarity of a node, and because its degree is required, we test three forms of assessing it: (i) considering only the incoming connections; (ii) only the outgoing connections; (iii) both incoming and outgoing connections. From these three possibilities, the second has proved to be the best and, for that, in all subsequent tests where topological similarity is used, we calculate it using the outgoing node connections to compute its degree.

Focusing on the node similarity we establish that, in order for two nodes to be compared, a dissimilarity matrix indicating the cost of aligning each terminal, non-terminal and parameter has to be defined, so that the method is easily generalisable. Results show that in the majority of tests, performing crossover using node similarity outperforms the standard crossover (i.e., no alignment is used).

Then, after analysing both the topological and node similarities, in isolation, we perform tests using both similarities to compute the cost matrix. Results demonstrate that this option is the one that yields better results.

Finally, and taking into account the previous results, we replace the standard crossover with the version with alignment (considering topological and node similarities) and perform evolutionary runs. Results show that crossover with alignment yields better results than the standard one.

Chapter 6

Families

Typically, when we observe a set of works from a given artist, or artistic movement, we naturally classify them as belonging to the same class, genre, or style. Often, the collection is more interesting than the individual works, revealing more information about the artistic goals, intentions and aesthetics of the author. To some extent, the work of artists implies creating a visual language and expressing themselves using the power, and constraints of that language. As the research of Stiny and Gips [78] on Shape Grammars (Section 2.3) demonstrates, even when this language is not explicitly defined by the author, in some cases it is possible to derive and formally express the rules that capture the underlying principles of a set of artifacts (e.g. Frank Lloyd Wright's prairie houses), and then use this grammar to create new instances that are consistent with the author's artistic practice. This observation is the main motivation for this chapter.

An additional motivation comes from the following observation, based on our previous experiments on the evolution of images: often, when we look at an evolved population, we find that it is more interesting as a whole than the images it contains when observed in isolation. This can be explained as follows; an evolved population tends to be composed of images that share a common genetic background, their genotypes tend to be similar and, as such, the images they give rise to tend to share several visual characteristics. Therefore, each image is perceived in a context which is supplied by the others and, as a whole, as variations on the same theme[1].

In the following sections, a scheme to evaluate families is going to be proposed (Section 6.1) and tested (Section 6.2). Whereas previous works on GP tend to ignore the characteristics among a set of phenotypes, we consider: the quality of each one;

[1]Some spurious images tend to exist.

the differences of quality among them; the consistency of the set; the diversity of the set.

Important to mention that the work described in this chapter has been submitted and accepted as a demo paper in the International Joint Conference on Artificial Intelligence 2015 (Appendix E.3).

6.1 Evaluating Families

As previously mentioned, we are primarily interested in the evolution of families. In the present scenario, families derive from grammars and from their non-deterministic nature. From that, it is clear that mapping them multiple times to phenotypes, using different seeds, may lead to different outputs, forming the family. To that end, we developed a fitness function that takes into account several aspects of each individual of the family and of the family as a whole. The principles that guided the development of this formula are:

1. The quality of each individual belonging to the family should be maximised;

2. The differences in quality should be minimised;

We consider that these are necessary conditions, since a collection of individuals that are deprived of interest on their own or a collection composed of extremely good and extremely bad ones cannot be considered of high quality. These conditions are not, however, sufficient. For instance, a collection composed exclusively of the same high quality individuals would meet those two criteria, but it could hardly be found interesting. Thus, we must take into account the relations between the elements of the collection:

3. A proper degree of diversity should exist.

Thus, the set of individuals should be diverse, to avoid monotony, but, at the same time, they should share some similarities, otherwise they would no longer be intuitively classified as belonging to the same family. The way these principles were translated into a fitness function and computationally implemented is described in the following paragraphs.

Being S a set of ordered I phenotypes, which belong to the same family, we begin by calculating $fit_{ind}(I), \forall I \in S$, allowing us to compute the mean quality of the set, $\overline{fit_{ind}}$, and the standard deviation of quality, $\sigma_{fit_{ind}}$, thus addressing the first two principles we have enunciated.

Parameters	Values
Number of runs	30
Number of generations	100
Population size	100
Crossover probability	0.6
Mutation probability	0.1
Tournament size	10
Elite size	Top 2%
Maximum expansion steps	100000
$\|S\|$	10
μ	$\{0.1, 0.3, 0.5, 0.7\}$
σ	0.2
a, b	$\{(1,1), (3,1), (1,3)\}$

Table 6.1: Parameters used in the experiments with families of CFDGs renderings.

To address the third principle, we calculate the similarity among all pairs of individuals belonging to the sample and its average. This raises a major difficulty because, for many domains, finding a suitable similarity metric is an open problem.

Taking all the previous into account, it is then possible to propose a fitness function for the assessment of families that addresses the three principles we enunciated, as follows:

$$f(S) = log\left(1 + \frac{\overline{fit_{ind}}}{1 + \sigma_{fit_{ind}}}\right)^{a} \times log(1 + N(sim(S), \mu, \delta))^{b}, \tag{6.1}$$

where N is the normal distribution function and sim the similarity of a set of phenotypes, which yields values in the $[0, 1]$ interval. By establishing different values for μ we can adjust the desired degree of similarity, and by setting different σ values we adjust the penalisation for deviating from that desired similarity. In this way, the use of the normal distribution rewards individuals which produce sets of phenotypes whose mean similarity is close to μ. The log function prevents evolution from focusing exclusively in one of the components of the formula. Finally, the exponents a and b allow us to adjust the importance given to each component.

6.2 Experimentation

To assess the validity of the proposed fitness function and the ability of the evolutionary engine to maximise it we conducted a wide variety of tests using the graph-based evolutionary engine (Section 3.1) to evolve families of images, which are the outcome of multiple renderings of individuals resembling CFDGs. The sample size, $|S|$, was

set to 10, meaning that each genotype is rendered 10 times using different rendering seeds. Table 6.1 summarises the parameter settings used in the course of the experiments.

As in some experiments during previous sections, we used a combination of "aesthetic measures" adapted from evolutionary art literature to evolve CFDGs. We resort to one of these combinations to assess the quality, fit_{ind}, of each image, I, in the sample, S. This particular combination focuses on the chromatic characteristics of the image, and uses two aesthetic measures: *Bell Curve* and *Contrasting Colours*, as following:

$$fit_{ind}(I) = log(1 + bell(I)) \times log(1 + cont\_colours(I)). \tag{6.2}$$

For simplicity reasons, and because the aim of this section is only to assess the validity of the proposed fitness function, we chose to estimate the similarity of individuals (in this case, images) on a pixel by pixel basis, calculating the root mean square error, $rmse$, over the three channels of the RGB colour space among pairs of images. R, G and B values are scaled to $[0, 1]$ prior to this calculation. Therefore, for the purpose of this study, the similarity of the set of samples S is given by the following formula:

$$sim(S) = \frac{\sum_{i=1}^{|S|-1} \sum_{j>i+1}^{|S|} (1 - rmse(S_i, S_j))}{\frac{|S| \times (|S|-1)}{2}} \tag{6.3}$$

We will now move into the designed experiments, focusing the importance of each part of the proposed fitness assignment scheme.

As a first step, we conducted experiments with $b = 0$, thus ignoring the similarity among the images of S. As expected, these runs typically converged to families composed of repetitions of the same image or minor variations of it. In some cases, however, the algorithm was able to find visual families composed of diversified images. Figure 6.1 illustrates these three types of results. The main conclusion is that ignoring the similarity among images implies having no control over the diversity of the family, which often leads to disappointing results due to lack or excess of diversity.

We also conducted experiments where we only took into account the diversity of the images (i.e., $a = 0$). As predictable, the results of these runs are poor. In the considered experimental conditions, and without any pressure to evolve images of high quality, it is trivial to match any given target similarity value using simple shapes (see Figure 6.2).

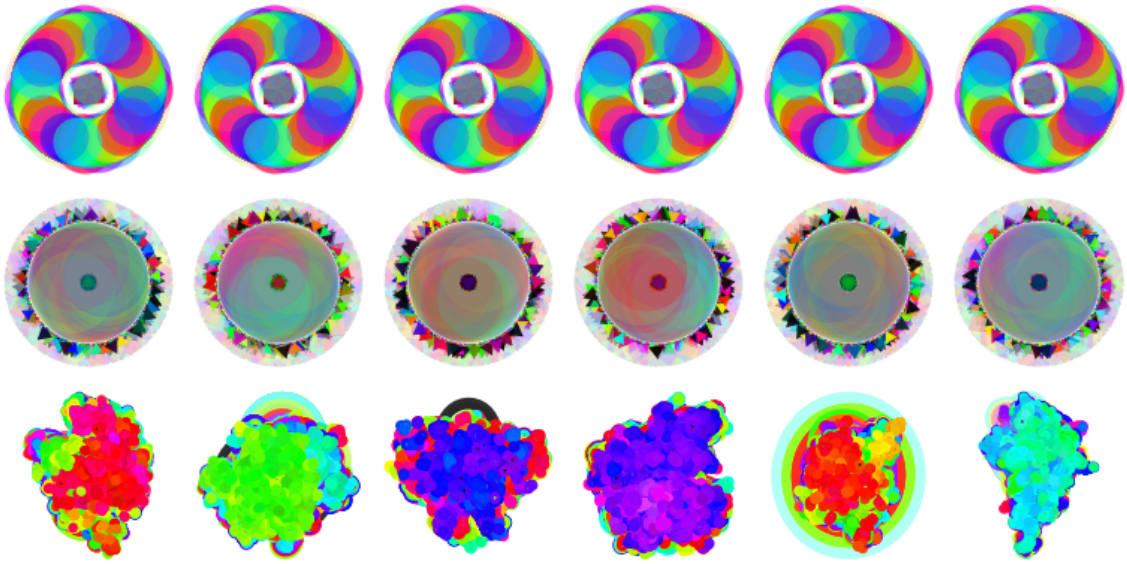


Figure 6.1: Samples of the fittest individuals from three independent runs with $b = 0$. Each line presents samples of images produced by a single individual.

Figure 6.2: Samples of the fittest individual of an evolutionary run with $a = 0$, $\mu = 0.7$ and $\sigma = 0.2$.

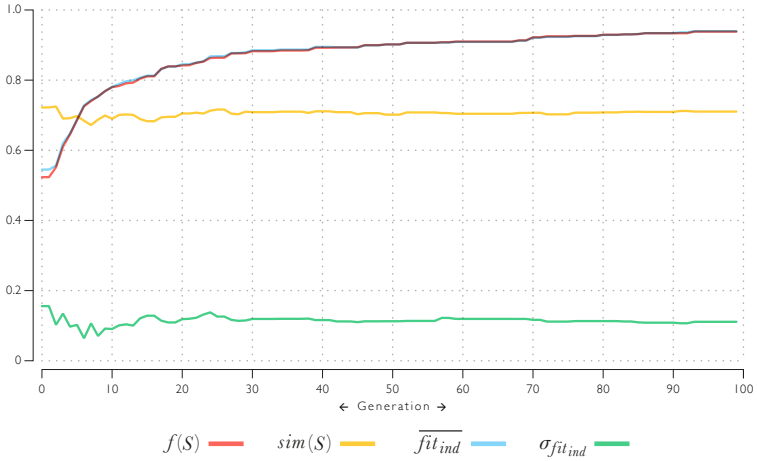


Figure 6.3: Evolution of the $f(s)$, $\overline{fit_{ind}}$, $\sigma_{fit_{ind}}$ and $sim(S)$ of the best individual when $\mu = 0.7$ and $a = b = 1$. Results are averages of 30 independent runs and have been normalised to improve readability.

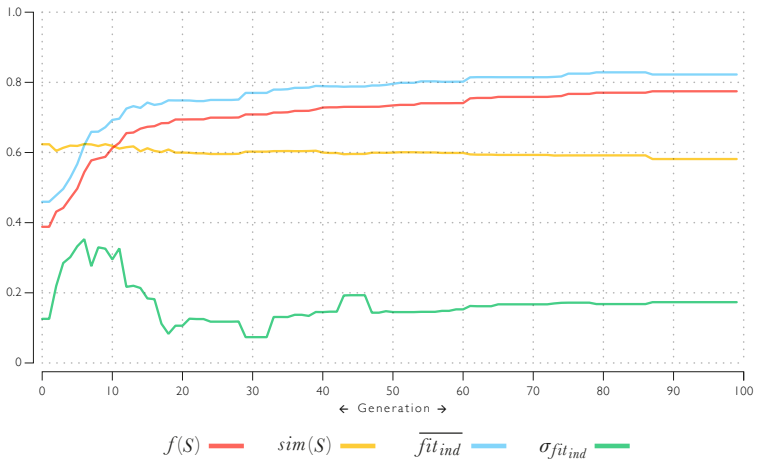


Figure 6.4: Evolution of the $f(s)$, $\overline{fit_{ind}}$, $\sigma_{fit_{ind}}$ and $sim(S)$ of the best individual when $\mu = 0.5$ and $a = b = 1$. Results are averages of 30 independent runs and have been normalised to improve readability.

Based on these preliminary results, which in essence, confirm that the assessment of the quality of a family requires taking into account the quality of the images as well as their diversity, we conducted runs where both these aspects were considered (i.e., $a \neq 0 \wedge b \neq 0$). Figures 6.3, 6.4, 6.5, 6.6 summarise the results obtained in these runs when using μ values of 0.7, 0.5, 0.3 and 0.1, respectively, with $a = b = 1$. They present the evolution of the fitness of the best individual, $f(S)$, as well as the corresponding $\overline{fit_{ind}}$, $\sigma_{fit_{ind}}$ and $sim(S)$. To promote readability, all values except those of $sim(S)$, that are already in the $[0, 1]$ interval, have been normalised by dividing the raw value by the maximum value found in the course of the experiments. The results of each figure are averages of 30 independent runs.

A brief perusal of the results and comparison among charts reveals that as the target similarity, μ, decreases, the difficulty of the task increases since, in general, lower targets for μ lead to lower $f(S)$ throughout the entire course of the runs. This confirms an intuitive idea: it is harder to evolve a family composed of high quality images that are dissimilar than one composed of high quality similar images.

Focusing on the results depicted in Figure 6.3, one can observe that the target similarity value of 0.7 is quickly reached. As so, the increase in $f(S)$ results, mainly, from increases on the average individual quality of the images of the sample, $\overline{fit_{ind}}$. After some initial fluctuations, the standard deviation of quality, $\sigma_{fit_{ind}}$, appears to decrease at a very slow rate. Comparing this behaviour with the one observed when $\mu = 0.5$, depicted in Figure 6.4, reveals the influence of this parameter and how it affects the difficulty of the task: the value reached by $f(S)$ is lower than the one

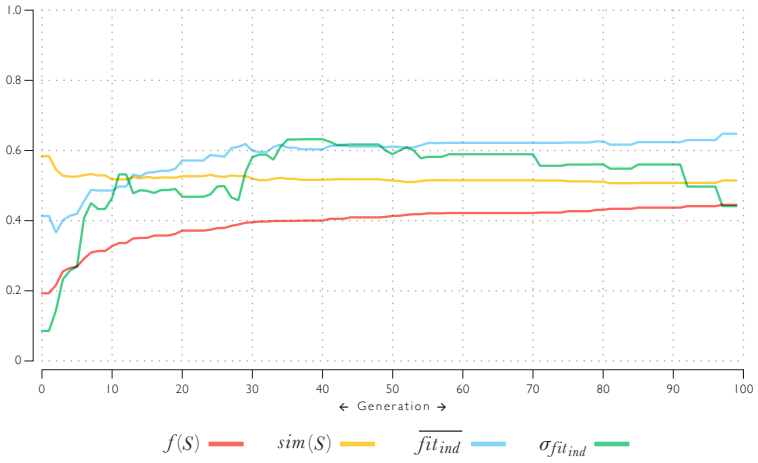


Figure 6.5: Evolution of the $f(s)$, $\overline{fit_{ind}}$, $\sigma_{fit_{ind}}$ and $sim(S)$ of the best individual when $\mu = 0.3$ and $a = b = 1$. Results are averages of 30 independent runs and have been normalised to improve readability.

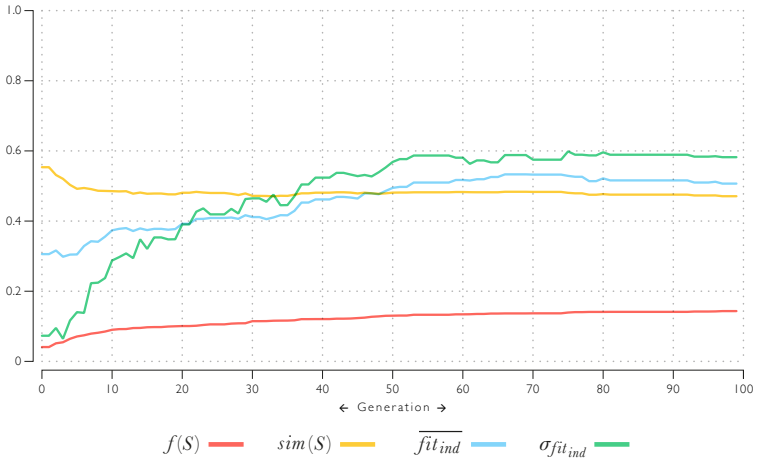


Figure 6.6: Evolution of the $f(s)$, $\overline{fit_{ind}}$, $\sigma_{fit_{ind}}$ and $sim(S)$ of the best individual when $\mu = 0.1$ and $a = b = 1$. Results are averages of 30 independent runs and have been normalised to improve readability.

obtained with $\mu = 0.7$ and, even though $sim(S)$ gradually decreases throughout the runs, the target similarity value is never reached. This quest for diversity affects other components of fitness; as it can be observed, after some initial fluctuations, $\sigma_{fit_{ind}}$ steadily rises, which is an undesired side effect. In other words, although the average quality of the sampled images is still steadily increasing, the consistency of their quality is being sacrificed to serve diversity. Interestingly, as is the case of $\mu = 0.7$, the lines for $f(S)$ and $\overline{fit_{ind}}$ follow similar paths throughout the runs, which may indicate that individual image quality exerts a dominant evolutionary pressure in these circumstances.

The results depicted in Figure 6.5, concerning the runs with $\mu = 0.3$, reveal the same overall trend as the ones obtained for $\mu = 0.5$, in the sense that maximising $f(S)$ is becoming significantly harder, the target μ value is not reached, and consistent quality is being sacrificed to attain diversity. However, the behaviour is more erratic, which highlights the tension between contradictory evolutionary pressures.

The analysis of the results obtained for $\mu = 0.1$ (Figure 6.6) shows that the evolutionary algorithm was unable to find solutions that are a good compromise between the different components of $f(S)$. In this case, the target μ value is not reached, $\sigma_{fit_{ind}}$ increases throughout most of the run, and although average quality increases during the first half of the runs, then it stabilises and decreases during the second half. Thus, in these circumstances, the quest for diversity appears to overpower other evolutionary pressures, and maximising $f(S)$ becomes extremely difficult.

The analysis of the visual results obtained in the course of the experiments unavoidably entails some degree of subjectivity. Furthermore, it is impossible to present all the visual families that were evolved. As such, rather than making a thorough analysis of the visual outcomes, or showing the families that we prefer, we focus on presenting to the reader results that are representative and that can be expected with different experimental settings. For each μ setting we randomly select 1 out of the 30 evolutionary runs conducted. We then pick the fittest individual found in each of these runs and present samples of the family it defines. These are presented in Figures 6.7, 6.8, 6.9, 6.10, which correspond to runs with $\mu = 0.7$, 0.5, 0.3 and 0.1, respectively. More examples of evolved families can be found in Appendix D.

Analysing the visual results obtained with $\mu = 0.7$, Figure 6.7, one can observe that: the colourful nature of the images and the colour gradients they exhibit reflect the aesthetic measures used to assess the individual quality of the images; all the samples produced by an individual share the same visual characteristics, and we consider that it is safe to state that one would naturally perceive these images as belonging to the same family. Unlike the results obtained when diversity is ignored (see Figure 6.1) there is some degree of diversity among the samples, which is also desirable. The same statements can be made for the results obtained with $\mu = 0.5$ (Figure 6.8). Comparing the visual results obtained with these two settings we can also observe that the diversity of the samples of the visual families evolved with $\mu = 0.5$ is higher than the ones evolved with $\mu = 0.7$. In our subjective opinion, we tend to prefer the sets of images evolved with $\mu = 0.5$, since we consider that they can consistently produce diversity without a significant loss of image quality. This aspect becomes more visible as the number of samples per family increases.

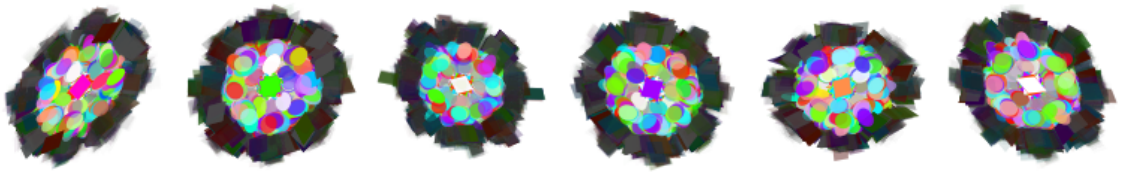


Figure 6.7: Sample of the fittest individual from a run with $\mu = 0.7$ and $a = b = 1$.

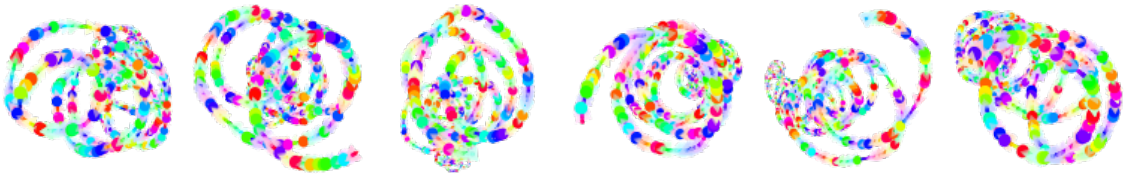


Figure 6.8: Sample of the fittest individual from a run with $\mu = 0.5$ and $a = b = 1$.

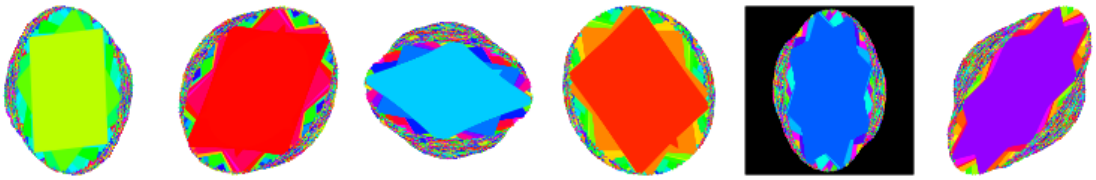


Figure 6.9: Sample of the fittest individual from a run with $\mu = 0.3$ and $a = b = 1$.

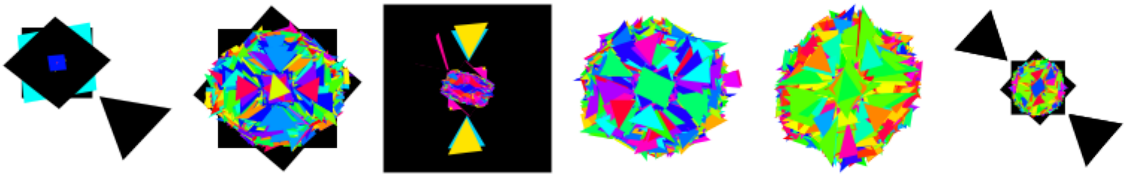


Figure 6.10: Sample of the fittest individual from a run with $\mu = 0.1$ and $a = b = 1$.

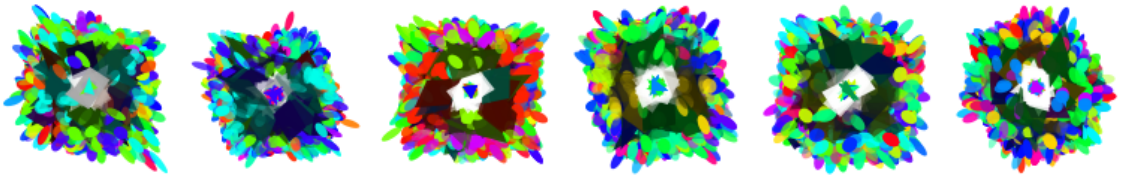


Figure 6.11: Sample of the fittest individual from a run with $\mu = 0.7$, $a = 3$ and $b = 1$.

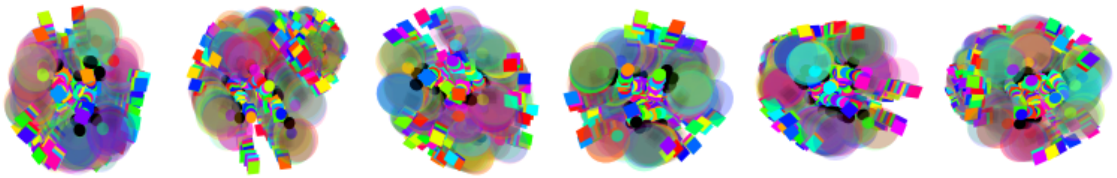


Figure 6.12: Sample of the fittest individual from a run with $\mu = 0.7$, $a = 1$ and $b = 3$.

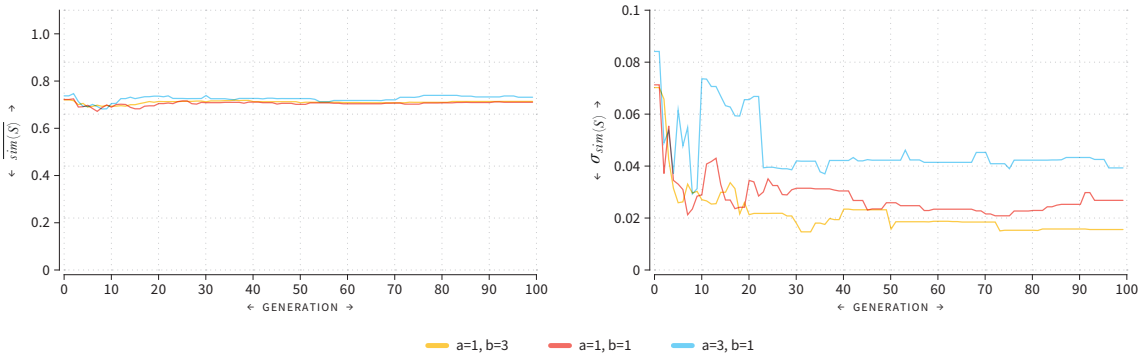


Figure 6.13: Evolution of the $sim(S)$ of the best individual when $\mu = 0.7$ and $(a, b) \in \{(1, 1), (3, 1), (1, 3)\}$. Results are averages of 30 independent runs and have been normalised to improve readability. On the left it is depicted the evolution of the average of the best individual and on the right the standard deviation.

An analysis of the visual results obtained with $\mu = 0.3$ reveals the difficulties of the evolutionary algorithm to find a compromise between the different aspects of fitness and, as such, to maximise $f(S)$. As we can observe in Figure 6.9, some spurious images exist, that could hardly be classified as belonging to the same image family as the others. More often than not, these images tend to be composed of a few simple shapes. In other words, diversity is being attained through the generation of images that, although different, are of very low quality (according to the considered aesthetic measures). These visual results provide insight regarding the behaviour of $\sigma_{fit_{ind}}$ during the course of these runs (see Figure 6.5), as average image quality increases the generation of these images of extreme low quality makes $\sigma_{fit_{ind}}$ increase. Furthermore, if we manually remove these spurious images from the samples, the diversity values become similar to those obtained with $\mu = 0.5$. Thus, to some extent, the evolutionary algorithm is not increasing the diversity of the set, at least in the manner that we wished, since it is increasing diversity by generating images that do not belong to the same family. The visual results obtained with $\mu = 0.1$ depict, and exaggerate the same tendencies.

Overall, the visual results confirm, and help to explain, the quantitative results obtained in the course of the experiments, providing additional clues to the explanation of the observed behaviour.

We will now focus on the results achieved while varying the weights a and b of the proposed fitness function. During this tests we used $\mu = 0.7$ and $\sigma = 0.2$.

Figure 6.13 depicts the evolution of the similarity for $(a, b) \in \{(1, 1), (3, 1), (1, 3)\}$. As expected, incrementing the weight of the quality part of the fitness function, i.e.,

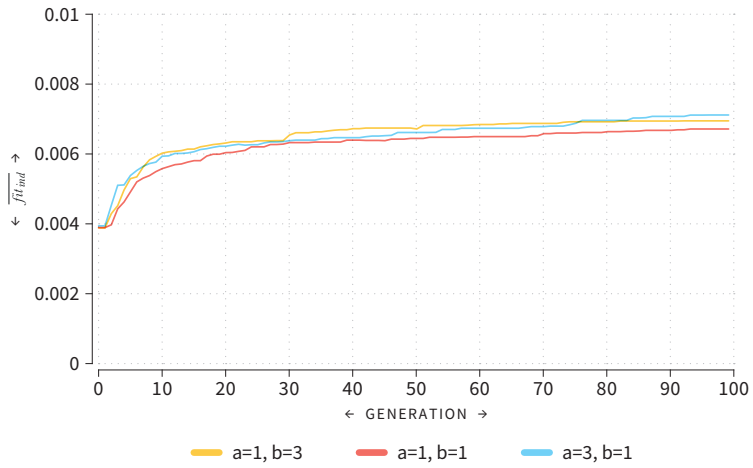


Figure 6.14: Evolution of the $\overline{fit_{ind}}$ of the best individual when $\mu = 0.7$ and $(a, b) \in \{(1, 1), (3, 1), (1, 3)\}$. Results are averages of 30 runs.

$a = 3$ and $b = 1$, the evolution struggles to reach the target similarity ($\mu = 0.7$) because the individual's quality is what is leading evolution. On the other hand, if preference is set towards the similarity of the set which defines the family ($a = 1$, $b = 3$) results get near those reached by the experiment where both components have the same weight ($a = b = 1$) in what concerns the average of the similarity of the best individuals. Although, if looking at the standard deviation of the similarity, it is clear that the experiment where $a = 1$ and $b = 3$ surpasses those where the weight is the same, which leads to the conclusion that families evolved giving a higher weight to the similarity component tend to be closer to the similarity target, μ, than those evolved with $a = b = 1$.

Considering the evolution of the $\overline{fit_{ind}}$ for the same testing scenarios, Figure 6.14, it is possible to observe that both weight variations provide better results than those presented by $a = b = 1$. When increasing the pressure of the quality part ($a = 3$, $b = 1$) the results are easily perceived as an outcome of this rise in the weight of a, and consequently to a bigger pressure to evolve high quality individuals. Although, the same does not happen when $b = 3$; here, the fact that the results are better than those produced by $a = b = 1$ is explained after the fact that an increase in the similarity component leads to individuals that are more closer to the target similarity value, μ. Moreover, this individuals due to the first part of the fitness scheme, tend to be of high individual quality and, as in this case, μ is set to a high value ($\mu = 0.7$), images tend to be very similar and of high quality, promoting an increase in the quality of the family.

Visual results for the tests where the weights of a and b are varied can be found in Figures 6.11 and 6.12, where $a = 3$ and $b = 3$, respectively. If more results are pretended, refer to Appendix D. As perceived in the results evolved with $a = 3$ and $b = 1$, families tend to have smoother transitions between colours than those evolved with $a = b = 1$, which is a result of higher fitness values, namely from the *Bell* component of the used combined fitness function. Results from $a = 1$ and $b = 3$ have no clear distinction from those evolved with $a = b = 1$ (with respect to similarity) which is explained by what had already been mentioned before; similarity is easy to achieve.

6.3 Conclusions

We present a novel approach for the evolution and assessment of families. Unlike previous efforts, we assign fitness based on the properties of the generated set of individuals and their diversity. The proposed fitness function promotes the maximisation of the individual quality, the minimisation of differences in quality, and a given degree of diversity among the individuals of the same family.

After proving that without taking into account similarity it is impossible to assure that families are composed of individuals that are not repetitions of the same pattern or minor variations of it, and that, if considering only the similarity the outputs are not the expected ones due to their lack of quality, we moved into varying the target similarity.

Subjectively, we have successfully achieved an important goal; the families that result from the evolution with an appropriate fitness function genuinely have a family-like appearance and are of great interest. Moreover, when the weight of both components of the fitness function is the same it is proven that, in the majority of the cases, the results can be seen as families. Although, if the similarity target is set to a low value (such, as $\mu = 0.3$ or $\mu = 0.1$) the similarity component is attained by evolving families where some of the elements are very simple and, as so, lower the average similarity of the set.

Changing the weights a and b has proven to produce the expected results if increasing the one related with the quality part of the individuals. The same cannot be stated when varying b; that can be explained by the ease with what a target similarity is reached.

Chapter 7

Conclusions and Future Work

Aiming at briefly describing all the work developed during the course of this Dissertation we will, in Section 7.1, start by detailing how work was planned, dividing it into several tasks. Then, in Section 7.2, the summary of the results of each one of the tasks will be conducted. To end, focus is given to future work (Section 7.3).

7.1 Work Planning

The planning of the present Dissertation has been divided into two main stages: completion and expansion of previous work.

During the first term, we intended to search and write about the state of the art surrounding the creation of an evolutionary engine capable of evolving generative grammars, using graph-based Genetic Programming and grammatical formulations. After accomplishing that, refinements to the previously developed engine were done, testing it against distinct automated fitness assignment schemas.

The above engine was developed only taking into account the evolution of Context Free Design Grammars [11]. As so, in the second term, efforts to generalise it, turning the adaptation to other domains easier were investigated and implemented. In an attempt to improve the operators used by the proposed evolutionary algorithm, methodologies which are structure aware were also tried.

Simplifying, we can divide the developed work in 9 tasks, which are now presented:

1. Revision of the bibliography and survey of the state of the art;
2. Refinement of the graph based evolutionary engine for generative grammars;
3. Development of automated fitness assignment schemes;

4. Writing of intermediate report and refinement of the work plan for the second semester;

5. Development of a grammatical evolution engine for generative grammars;

6. Development of semantic and structure aware methods for generative grammars;

7. Development of a fitness function capable of assessing the quality of a set of individuals;

8. Experimentation;

9. Dissemination: Dissertation and Papers.

7.2 Summary

Beginning with the State of the Art (Chapter 2), we detailed and searched approaches related with two different but related fields. First of all we referred to Evolutionary Computation and Algorithms, where focus was given mainly over Genetic Programming approaches. Later, a section focusing Grammars intuitively emerges, where the revision of several techniques aiming at grammar evolution is accomplished. Upon absorbing all the previously mentioned literature, and aiming at developing an evolutionary engine capable of successfully evolving solutions in different environments we raised the following questions, that will be answered during the next paragraphs:

1. Is it possible to come up with a tool capable of dealing with multiple problem environments and domains?

2. Can grammar formulations help in this task? If so, how?

3. Is it worth trying to combine multiple approaches from distinct types of methodologies?

In order to gain some insights that may lead us to be able to provide answers to the above questions we developed two evolutionary approaches (Chapter 3), with different forms of representation and, as so, with different operators (crossover, mutation and population initialisation). Starting with the graph-based approach, as the name indicates, individuals resemble a graph structure, where each node stands for a production rule and connections between them indicate flow of control, which can or

not pass parameters within calls. In this way, the graph itself represents a grammar, where the set of all nodes and links defines the system of production rules. In the other hand, a tree-based representation was used. Individuals represent derivation-trees from a user-provided pre-grammar, which is no more than a BNF with the addition of coding blocks.

Chapter 4 describes a set of comprehensive experiments over the two previous mentioned forms of representation. We tested them in two different domains. First, we evolved Context Free Design Grammars that were ultimately rendered into PNG images. We begin, in this scenario, by running tests with the tree and graph-based approaches. Both have proven to be capable of promoting the evolution of several distinct automatic fitness functions, in isolation and combined. From this point we tried to merge components from one and the other representation to see if that would lead to an increase in the performance of the evolutionary algorithm. As such, limiting the population initialisation to the one used for the creation of random derivation trees (for generalisation purposes), we test several combinations of mutation and crossover. Graph-mutation has proven to lead to the worse results. All the remaining setups where tree-mutation is used have shown no significant statistical difference. At this point we designed experiments using both crossover operators in the same evolutionary algorithm, e.g. tree-mutation with tree and graph crossover, with probabilities of 70% and 30%, respectively. Results show no clear difference in what regards the fitness of the best produced individuals. However, focusing the average fitness at a population level, a trend towards the tree mutation exits, i.e., increasing the rate of tree-crossover leads to higher fitnesses at a population level, which is an outcome of tree-crossover having a higher constructive rate than the graph one. For all this, we point the combination of tree-mutation with tree-crossover as apparently performing better than the ones where graph-crossover is employed.

Apart from the experiments where CFDGs were evolved, we have also tried to promote the evolution of grammars resembling musical sequences, that are later mapped into MIDI files, to ease reproduction. This part of the experimentation had the objective of working as a proof of concept for the capacity of the evolutionary engine to cope with different domains. As so, we used an evolutionary algorithm that uses operators from both representations. Because it is just a proof of concept, instead of an automatic fitness evaluation procedure we used a user-guided one. Results show that the evolutionary engine can promote the evolution of aesthetic results and, as so, that it has sufficient generalisation properties, answering this way to the first and second questions. That is, it is possible to develop an evolutionary engine capable

of addressing different tasks, which is facilitated by having a system where only the pre-grammar has to be changed, in order to produce the pretended results.

As the reader may remember, we have above mentioned that the graph-crossover operator was not providing results that were considered as good as the tree-crossover ones. Therefore, we investigated ways to improve it. In Chapter 5, alignment of structures is addressed. Graph-crossover, as opposed to tree-crossover, chooses the crossover cutting points at random locations. As such, the genetic material that is swapped can be completely incompatible. Alignment provides a one-to-one mapping from the smallest (H) to the largest graph (G), returning a list of nodes, where each pair (v, u) of nodes, $v \in H$ and $u \in G$, has the lowest alignment cost, i.e., they are similar. The computation of the cost matrix in alignment is made of two components, topological and node similarities. The first analysis the connections established between nodes, whereas the second takes a most semantic approach to similarity. Obtained results show that the computation of the cost matrix yields better results when using both components. Furthermore, it has also been proven that the alignment introduces improvements in the performance of the graph-crossover operator. It is then now possible to answer to the third question. As shown, in some circumstances, merging graph-crossover with tree-crossover leads to better results, proving that it may be worth combining components from several approaches.

Finally, in Chapter 6 we introduce a fitness function for the assessment of the quality of families of individuals. This is particularly interesting, as our main focus has been over the evolution of both images and music, which are encoded by non-deterministic grammars. That means that, if we perform their mapping to the phenotype (image / music) several times, providing different rendering seeds, the results may be distinct. As such, in our current scenario we do not actually have a family of individuals but a family of mappings of the same individual instead. The proposed fitness function promotes the maximisation of the individual quality, the minimisation of differences in quality, and a given degree of diversity among the individuals of the same family. We prove that all its components are important and that, together, they are capable of evolving sets of phenotypes that are easily identified as belonging to the same family.

7.3 Future Work

Concerning future work, there are several aspects that were not analysed, mainly due to the lack of time or because they were out of the scope of this Dissertation.

First of all, we could try applying automatic fitness evaluation procedures into the evolution of musical sequences. That would certainly speed up the process of evolution and, ultimately allow the reproduction of results. Still focusing the work done with music, the grammars that represent individuals, like CFDGs, are also non-deterministic. For that, the same rationale that was used for the evolution of families of images could be easily adapted to music, promoting the evolution of musical sequences that share a degree of similarity. Later, the multiple individuals that compose a family could be merged in order to form a single musical piece.

As one may remember, in experiments performed for the evolution of families of images, RMSE was used for computing the distance between the images. This metric is simplistic and, as such, other possibilities should be searched and tried in the future.

Alignment of individuals was just used to accomplish the choosing of crossover's cutting points in a more informed manner. However, what is in fact swapped between the two parents are subgraphs. For that, instead of using alignment to choose only the cutting points, we can try to also use it in the part of the crossover where a mapping between the two subgraphs, that are exchanged, is performed. Another aspect connected with the improvement of the evolutionary algorithm is related with the way that parents are selected. It would be interesting to, instead of applying the tournament selection based on fitness, try, for example, to base it on semantic similarity between individuals.

To end, and perhaps one of the most import points to be addressed in future work, the application of the graph-crossover operator and of the alignment rationale to Cartesian Genetic Programming, testing it then in classical optimisation problems, such as symbolic regression.

Bibliography

[1] International MIDI Association et al. Midi musical instrument digital interface specification 1.0. *Los Angeles*, 1983.

[2] Thomas Back, David B. Fogel, and Zbigniew Michalewicz. *Handbook of evolutionary computation*. IOP Publishing Ltd., 1997.

[3] Ellie Baker and Margo Seltzer. Evolving line drawings. In *ICGA*, page 627, 1993.

[4] Shumeet Baluja, Dean Pomerleau, and Todd Jochem. Towards automated artificial evolution for computer-generated images. *Connection Science*, 6(2-3):325–354, 1994.

[5] Wolfgang Banzhaf. Interactive evolution. *Evolutionary Computation*, 1:228–236, 2000.

[6] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies–a comprehensive introduction. *Natural computing*, 1(1):3–52, 2002.

[7] Jon Bird, Phil Husbands, Martin Perris, Bill Bigge, and Paul Brown. Implicit fitness functions for evolving a drawing robot. In *Applications of Evolutionary Computing*, pages 473–478. Springer, 2008.

[8] Jon Bird and Dustin Stokes. Minimal creativity, evaluation and fractal pattern discrimination. *Programme Committee and Reviewers*, page 121, 2007.

[9] Janet Clegg, James A. Walker, and Julian F. Miller. A new crossover technique for cartesian genetic programming. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1580–1587. ACM, 2007.

[10] Ernesto Costa and Anabela Simões. *Artificial Intelligence: foundatons and applications (in portuguese)*. FCA, 3rd edition, 2008.

[11] C. Coyne. Context Free Design Grammar. http://www.chriscoyne.com/cfdg/, last accessed in June 2015.

[12] coord. Cunha, Lúcio, coord. Takahashi, Ricardo, and coord. Antunes, Carlos Alberto Henggeler de Carvalho. *Manual de computação evolutiva e metaheurística*. Imprensa da Universidade de Coimbra, Coimbra, 2012 2012.

[13] Charles Darwin. *On the origins of species by means of natural selection*. 1859.

[14] Richard Dawkins. The blind watchmaker: Why the evidence of evolution reveals a universe without design, 1986.

[15] Eelco den Heijer and Agoston E. Eiben. Comparing aesthetic measures for evolutionary art. In *Applications of Evolutionary Computation*, pages 311–320. Springer, 2010.

[16] Christoph Döpmann. Survey on the graph alignment problem and a benchmark of suitable algorithms. 2013.

[17] Agoston E. Eiben, Elena Marchiori, and VA Valko. Evolutionary algorithms with on-the-fly population size adjustment. In *Parallel Problem Solving from Nature-PPSN VIII*, pages 41–50. Springer, 2004.

[18] Agoston E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer, Verlag, 2003.

[19] Yuval Fisher, editor. *Fractal Image Compression: Theory and Application*. Springer, London, 1995.

[20] David B. Fogel. Evolutionary programming: an introduction and some current directions. *Statistics and Computing*, 4(2):113–129, 1994.

[21] Steve De Furia and Joe Scacciaferro. *MIDI programmer's handbook*. IDG Books Worldwide, Inc., 1990.

[22] Philip Galanter. What is generative art? complexity theory as a context for art theory. In *In GA2003–6th Generative Art Conference*. Citeseer, 2003.

[23] Philip Galanter. The problem with evolutionary art is... In *Applications of Evolutionary Computation*, pages 321–330. Springer, 2010.

[24] Simon Harding, Vincent Graziano, Jürgen Leitner, and Jürgen Schmidhuber. Mt-cgp: Mixed type cartesian genetic programming. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, pages 751–758. ACM, 2012.

[25] Simon L. Harding, Julian F. Miller, and Wolfgang Banzhaf. Self-modifying cartesian genetic programming. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1021–1028. ACM, 2007.

[26] David A. Hart. Toward greater artistic control for interactive evolution of images and animation. In *Applications of evolutionary computing*, pages 527–536. Springer, 2007.

[27] John H. Holland. Genetic algorithms. *Scientific american*, 267(1):66–72, 1992.

[28] John E. Hopcroft. *Introduction to automata theory, languages, and computation*. Pearson Education India, 1979.

[29] J. Horigan and M. Lentczner. Context Free. http://www.contextfreeart.org/, last accessed in September 2009.

[30] Compuserve Incorporated. GIF Graphics Interchange Format: A standard defining a mechanism for the storage and transmission of bitmap-based graphics information. Columbus, OH, USA, 1987.

[31] A. Karperien. Fraclac for imagej. In *http://rsb.info.nih.gov/ij/plugins/fraclac/FLHelp/Introduction.htm*, 1999-2013.

[32] John R. Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[33] Oleksii Kuchaiev, Tijana Milenković, Vesna Memišević, Wayne Hayes, and Nataša Pržulj. Topological network alignment uncovers biological function and phylogeny. *Journal of the Royal Society Interface*, 7(50):1341–1354, 2010.

[34] Matthew Lewis. Evolutionary visual art and design. In *The Art of Artificial Evolution*, pages 3–37. Springer, 2008.

[35] Peter Linz. *An introduction to formal languages and automata*. Jones & Bartlett Publishers, 2011.

[36] Heikki Maaranen, Kaisa Miettinen, and Antti Penttinen. On initial populations of a genetic algorithm for continuous optimization problems. *Journal of Global Optimization*, 37(3):405–436, 2007.

[37] P. Machado and A. Cardoso. All the truth about NEvAr. *Applied Intelligence, Special Issue on Creative Systems*, 16(2):101–119, 2002.

[38] Penousal Machado. *Artificial Intelligence and Art (in portuguese)*. PhD thesis, University of Coimbra, Coimbra (Portugal), 2006.

[39] Penousal Machado and Amílcar Cardoso. Computing aesthetics. In *Advances in Artificial Intelligence*, pages 219–228. Springer, 1998.

[40] Penousal Machado, João Correia, and Filipe Assunção. Graph-Based Evolutionary Art. In Amir Gandomi, Amir Hossein Alavi, and Conor Ryan, editors, *Handbook of Genetic Programming Applications*. Springer, Berlin, 2015.

[41] Penousal Machado and Henrique Nunes. A step towards the evolution of visual languages. In *First International Conference on Computational Creativity, Lisbon, Portugal*, 2010.

[42] Penousal Machado, Henrique Nunes, and Juan Romero. Graph-based evolution of visual languages. In *Applications of Evolutionary Computation*, pages 271–280. Springer, 2010.

[43] Penousal Machado, Juan Romero, Amílcar Cardoso, and Antonio Santos. Partially interactive evolutionary artists. *New Generation Computing – Special Issue on Interactive Evolutionary Computation*, 23(42):143–155, 2005.

[44] Penousal Machado, Juan Romero, and Bill Manaris. Experiments in computational aesthetics: an iterative approach to stylistic change in evolutionary art. In Juan Romero and Penousal Machado, editors, *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*, pages 381–415. Springer Berlin Heidelberg, 2007.

[45] Krešimir Matković, László Neumann, Attila Neumann, Thomas Psik, and Werner Purgathofer. Global contrast factor-a new approach to image contrast. In *Proceedings of the First Eurographics conference on Computational Aesthetics in Graphics, Visualization and Imaging*, pages 159–167. Eurographics Association, 2005.

[46] Jon McCormack. Interactive evolution of l-system grammars for computer graphics modelling. *Complex Systems: from biology to computation*, pages 118–130, 1993.

[47] Jon McCormack. New challenges for evolutionary music and art. *ACM SIGEVOlution*, 1(1):5–11, 2006.

[48] Tijana Milenkoviæ and Nataša Pržulj. Uncovering biological network function via graphlet degree signatures. *Cancer informatics*, 6:257, 2008.

[49] Julian F. Miller. Cartesian genetic programming. In Julian F. Miller, editor, *Cartesian Genetic Programming*, Natural Computing Series, pages 17–34. Springer Berlin Heidelberg, 2011.

[50] Julian F. Miller and Stephen L. Smith. Redundancy and computational efficiency in cartesian genetic programming. *Evolutionary Computation, IEEE Transactions on*, 10(2):167–174, 2006.

[51] Toshio Mori, Yoshimichi Endou, and Akira Nakayama. Fractal analysis and aesthetic evaluation of geometrically overlapping patterns. *Textile research journal*, 66(9):581–586, 1996.

[52] Ronald W. Morrison and Kenneth A. De Jong. Measurement of population diversity. In *Artificial Evolution*, pages 31–41. Springer, 2002.

[53] Miguel Nicolau and Ian Dempsey. Introducing grammar based extensions for grammatical evolution. In *IEEE Congress on Evolutionary Computation*, pages 648–655, 2006.

[54] Michael O'Neill, James McDermott, John Mark Swafford, Jonathan Byrne, Erik Hemberg, Anthony Brabazon, Elizabeth Shotton, Ciaran McNally, and Martin Hemberg. Evolutionary design using grammatical evolution and shape grammars: Designing a shelter. *International Journal of Design Engineering*, 3(1):4–24, 2010.

[55] Michael O'Neill and Conor Ryan. Grammar based function definition in grammatical evolution. In *GECCO*, pages 485–490, 2000.

[56] Michael O'Neill and Conor Ryan. *Grammatical evolution: evolutionary automatic programming in an arbitrary language*, volume 4. Springer, 2003.

[57] Michael O'Neill, Conor Ryan, Maarten Keijzer, and Mike Cattolico. Crossover in grammatical evolution. *Genetic programming and evolvable machines*, 4(1):67–93, 2003.

[58] Michael O'Neill, John Mark Swafford, James McDermott, Jonathan Byrne, Anthony Brabazon, Elizabeth Shotton, Ciaran McNally, and Martin Hemberg. Shape grammars and grammatical evolution for evolutionary design. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1035–1042. ACM, 2009.

[59] Michael O'Neill and Conor Ryan. Genetic code degeneracy: Implications for grammatical evolution and beyond. In *Advances in Artificial Life*, pages 149–153. Springer, 1999.

[60] Michael O'Neill and Conor Ryan. Grammatical evolution. In *Grammatical Evolution*, pages 33–47. Springer, 2003.

[61] Francisco B. Pereira, Penousal Machado, Ernesto Costa, and Amílcar Cardoso. Graph based crossover–a case study with the busy beaver problem. In *Proceedings of the 1999 Genetic and Evolutionary Computation Conference*, 1999.

[62] Riccardo Poli. Evolution of graph-like programs with parallel distributed genetic programming, 1997.

[63] Riccardo Poli. Parallel distributed genetic programming. Technical report, School of Computer Science, University of Birminham, 1999.

[64] Riccardo Poli and Stefano Cagnoni. Evolution of pseudo-colouring algorithms for image enhancement with interactive genetic programming. *Cognitive Science Research Papers - University of Birminham CSRP*, 1997.

[65] Riccardo Poli, William B. Langdon, Nicholas F. McPhee, and John R. Koza. *A field guide to genetic programming*. Lulu. com, 2008.

[66] Aristid Lindenmayer Przemyslaw Prusinkiewicz, Aristid Lindenmayer, James S. Hanan, F. David Fracchia, and Deborah Fowler. The algorithmic beauty of plants. 1990.

[67] Steven Rooke. The evolutionary art of steven rooke. `http://srooke.com/`. last accessed in December 2014.

[68] Brian J. Ross, William Ralph, and Zong Hai. Evolutionary image synthesis using a model of aesthetics. In Gary G. Yen, Simon M. Lucas, Gary Fogel, Graham Kendall, Ralf Salomon, Byoung-Tak Zhang, Carlos A. Coello Coello, and Thomas Philip Runarsson, editors, *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 1087–1094, Vancouver, BC, Canada, 16–21 July 2006. IEEE Press.

[69] Franz Rothlauf. *Representations for genetic and evolutionary algorithms*. Springer, 2006.

[70] Rob Saunders and Kazjon Grace. Teaching evolutionary design systems by extending "context free". In *Applications of Evolutionary Computing*, pages 591–596. Springer, 2009.

[71] Karl Sims. *Artificial evolution for computer graphics*, volume 25. ACM, 1991.

[72] William M. Spears, Kenneth A. De Jong, Thomas Bäck, David B. Fogel, and Hugo De Garis. An overview of evolutionary computation. In *Machine Learning: ECML-93*, pages 442–459. Springer, 1993.

[73] William M. Spears et al. Crossover or mutation? In *FOGA*, pages 221–237, 1992.

[74] Lee Spector and Adam Alpern. Criticism, culture, and the automatic generation of artworks. In *AAAI*, pages 3–8, 1994.

[75] Branka Spehar, Colin Clifford, Ben R. Newell, and Richard P. Taylor. Universal aesthetic of fractals. *Computers and Graphics*, 27(5):813–820, October 2003.

[76] M. Srinivas and Lalit M. Patnaik. Adaptive probabilities of crossover and mutation in genetic algorithms. *Systems, Man and Cybernetics, IEEE Transactions on*, 24(4):656–667, 1994.

[77] George Stiny. Introduction to shape and shape grammars. *Environment and planning B*, 7(3):343–351, 1980.

[78] George Stiny and James Gips. Shape grammars and the generative specification of painting and sculpture. In *IFIP Congress (2)*, pages 1460–1465, 1971.

[79] Nils Svangård and Peter Nordin. Automated aesthetic selection of evolutionary art by distance based classification of genomes and phenomes using the universal similarity metric. In *Applications of Evolutionary Computing*, pages 447–456. Springer, 2004.

[80] Hideyuki Takagi. Interactive evolutionary computation: Fusion of the capabilities of ec optimization and human evaluation. *Proceedings of the IEEE*, 89(9):1275–1296, 2001.

[81] Tatsuo Unemi. A design of multi-field user interface for simulated breeding. In *Proceedings of the third Asian Fuzzy Systems Symposium*, pages 489–494, 1998.

[82] Tatsuo Unemi. Sbart 2.4: an iec tool for creating 2d images, movies, and collage. In *Workshop on Genetic Algorithms in Visual Art and Music*, pages 21–23, 2000.

[83] James A. Walker and Julian F. Miller. The automatic acquisition, evolution and reuse of modules in cartesian genetic programming. *Evolutionary Computation, IEEE Transactions on*, 12(4):397–417, 2008.

[84] Peter A. Whigham. Inductive bias and genetic programming. 1995.

[85] Peter A. Whigham. Search bias, language bias and genetic programming. In *Proceedings of the First Annual Conference on Genetic Programming*, pages 230–237. MIT Press, 1996.

[86] Peter A. Whigham et al. Grammatically-based genetic programming. In *Proceedings of the workshop on genetic programming: from theory to real-world applications*, volume 16, pages 33–41. Citeseer, 1995.

[87] Mitchell Whitelaw. *Metacreation: art and artificial life*. Mit Press, 2004.

[88] L. World. Aesthetic selection: The evolutionary art of steven rooke. *Computer Graphics and Applications, IEEE*, 16(1):4, 1996.