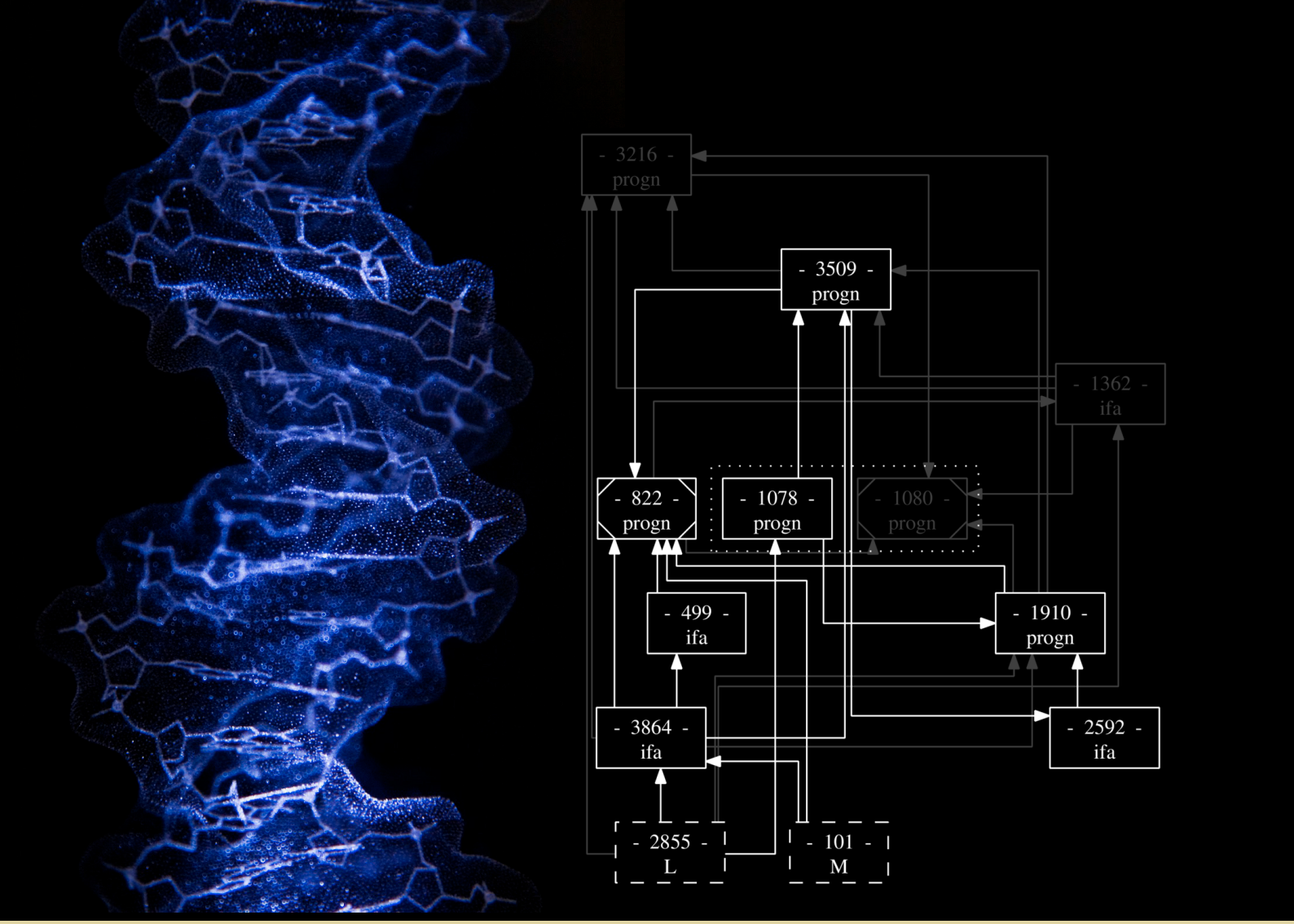


Rui Miguel Lourenço Lopes

A Computational Model Inspired by Gene Regulatory Networks

Doctoral thesis submitted to the Doctoral Program in Information Science and Technology, supervised by Full Professor Doutor Ernesto Jorge Fernandes Costa, and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

January 2015

· U C ·

Universidade de Coimbra

A Computational Model Inspired by Gene Regulatory Networks

A thesis submitted to the University of Coimbra
in partial fulfillment of the requirements for the
Doctoral Program in Information Science and Technology

by

Rui Miguel Lourenço Lopes
rmlopes@dei.uc.pt

Department of Informatics Engineering
Faculty of Sciences and Technology
University of Coimbra

Coimbra, January 2015

Financial support by Fundação para a Ciência e a Tecnologia,
through the PhD grant SFRH/BD/69106/2010.
A Computational Model Inspired by
Gene Regulatory Networks

ISBN 978-989-20-5460-5
Cover image: Composition of a Santa Fe trail program
evolved by ReNCoDe overlaid on the ancestors,
with a 3D model of the DNA crystal structure (by Paul Hakimata).


This dissertation was prepared under the supervision of

Ernesto J. F. Costa
Full Professor
of the Department of Informatics Engineering
of the Faculty of Sciences and Tecnology
of the University of Coimbra

In loving memory of my father.

Acknowledgements

Thank you very much to Prof. Ernesto Costa, for his vision on research and education, for all the support and contributions to my ideas, and for the many life stories that always lighten the mood. Thank you also to Nuno Lourenço for the many discussions, shared books, and his constant helpfulness. A general thank you to the Evolutionary and Complex Systems (ECoS) group for providing a friendly and enthusiast work environment. Also, a thank you to all the authors that sent us their articles, and to the open source community without whom the code that supports this thesis would not be possible.

A big thank you to my family for the unconditional support, and for providing countless reunions full of joy and spirit(s). In particular to my three women Celeste, Marta, and Irina, for all the support and care throughout this ride.

Resumo

Os algoritmos evolucionários (AE) são procedimentos de pesquisa estocástica em paralelo, vagamente inspirados pelos conceitos de selecção natural, genética, e hereditariedade. Estes têm sido aplicados com sucesso em muitos domínios e a Computação Evolucionária (CE) atrai hoje um número cada vez maior de investigadores das mais variadas áreas.

O final do século XX trouxe inúmeras descobertas na esfera biológica, permitidas pelas inovações tecnológicas subjacentes. Genomas completos foram sequenciados, incluindo o humano, e graças à crescente interdisciplinaridade dos investigadores sabe-se hoje que evolução é muito mais do que apenas selecção natural. Há também a influência do meio ambiente, a regulação genética, e o desenvolvimento. No cerne destes processos existe uma peça fundamental de maquinaria biológica, a Rede de Regulação Genética (Gene Regulatory Network - GRN). Esta rede resulta da interacção entre os genes e proteínas, bem como o meio ambiente, decidindo a expressão dos genes e, consequentemente, o desenvolvimento do organismo.

É reconhecido por vários investigadores que os conhecimentos biológicos têm avançado mais rápido do que a nossa capacidade de incorporá-los nos AEs, independentemente do facto de ainda estar por provar se é ou não benificial fazê-lo. Uma das principais críticas apontadas é que a abordagem à relação genótipo-fenótipo é diferente da observada na natureza. Um grande esforço tem sido feito por alguns investigadores para desenvolver novas representações, tendo alcançado não só melhores resultados em problemas de teste, mas também maior flexibilidade e aplicabilidade dos algoritmos. Para além disso, outros começaram recentemente a estudar computacionalmente o novo conhecimento da multiplicidade de mecanismos de regulação, que são fundamentais para ambos os processos de hereditariedade, e de desenvolvimento dos organismos, tentando incluir esses mecanismos nos AEs. No entanto, ainda são poucos os que são aplicados a problemas de aprendizagem e optimização, e muitos destes são normalmente desenvolvidos com um problema específico como alvo.

A principal contribuição desta tese é um modelo que integra redes de regulação artificiais como a representação genotípica de um sistema de Programação Genética (PG), e um algoritmo que mapeia estas redes em grafos de programas executáveis. Para além disso, variantes do modelo original foram também desenvolvidas, estendendo as capacidades do modelo às classes de problemas com definições recursivas, e com múltiplos resultados em paralelo. O modelo desenvolvido foi validado experimentalmente usando problemas de referência para sistemas de PG, desde regressão simbólica a controlo, e aprendizagem, com ou sem memória. As experiências realizadas permitiram avaliar a eficácia e a eficiência do algoritmo, bem como a influência de diferentes operadores e parameterizações. Apesar de algumas limitações que foram identificadas, a análise dos resultados mostra que este novo método é competitivo na maioria dos problemas testados, superando mesmo nalguns casos os resultados da literatura.

Abstract

Evolutionary Algorithms (EA) are parallel stochastic search procedures that are loosely inspired by the concepts of natural selection and genetic heredity. They have been successfully applied to many domains, and today Evolutionary Computation (EC) attracts a growing number of researchers from the most varied fields.

The end of the 20^{th} century brought uncountable discoveries in the biological realm, enabled by the underlying technological breakthroughs. Complete genomes have been sequenced, including the human one, and thanks to the increasing interdisciplinarity of researchers it is known today that there is much more to evolution than just natural selection, namely the influence of the environment, gene regulation, and development. At the core of these processes there is a fundamental piece of complex biological machinery, the Genetic Regulatory Network (GRN). This network results from the interaction amongst the genes and proteins, as well as the environment, governing gene expression and consequently the development of the organism.

It is a true fact that the biological knowledge has advanced faster than our ability to incorporate it into the EAs, despite of whether or not it is beneficial to do so. One of the main critics pointed-out is that the approach to the genotype-phenotype relationship is different from nature. A lot of effort has been put by some researchers into developing new representations, achieving not only improved benchmark results, but also extended flexibility and applicability of the algorithms. Moreover, others have recently started exploring computationally the new comprehension of the multitude of regulatory mechanisms that are fundamental in both the processes of inheritance and of development in natural systems, by trying to include those mechanisms in the EAs. However, few of these target machine learning problems, and most are usually developed with a specific problem domain in mind. The work presented here addresses this issue by incorporating a model of GRN in a Genetic Programming (GP) architecture.

This thesis main contribution is a model that incorporates Artificial Regulatory Networks as the genotypic representation in a GP-like system, and an algorithm that maps these networks into executable program graphs. Moreover, variants of the model were also developed, extending the capabilities of the approach to classes of problems with recursive definitions, and with multiple outputs. The efficacy and efficiency of this alternative were tested experimentally using typical benchmark problems for Genetic Programming systems, from regression to control, and logic design. Despite some limitations that were identified, the analysis of the results shows that this new method is competitive in most problem domains, even outperforming the state-of-the-art results in some cases.

Keywords

Computational Evolution, Artificial Genetic Regulatory Network, Developmental Systems, Evolutionary Strategies, Genetic Programming.

Contents

List of Tables

List of Figures

List of Algorithms

Listings

1

Introduction

It was at the very beginning of an introductory computer programming course that I first heard what is perhaps the oldest cliché in Computer Science (CS), that "computers are dumb, they can only do what you tell them to". Apparently as simple as an affirmation can be (and to me in that context it sounded like a valid statement), it points to the rather philosophical question that is the foundation of Artificial Intelligence (AI).

Even before the first AI conference [Russell and Norvig, 2010], Alan Turing proposed "The Imitation Game" (a.k.a "The Turing Test"), addressing the question of whether machines can or cannot think[1]. Few decades later, Marvin Minsky[2], one of the founders of the AI research field, also addressed the issue, stressing the point that computers can also manipulate symbols and that Turing and other researchers had written programs that tried to mirror human intellectual capabilities, using machines that were designed with only arithmetic operations as their goal. He finishes his article raising the question "Can computers do only what they are told?", and provides food for thought with an analogy with musical composition. Richard Dawkins[3] graciously points out that "The cliché is true only in the crashingly trivial sense, the same sense in which Shakespeare never wrote anything except what his first schoolteacher taught him to write – words".

[1] In *Computing Machinery and Intelligence*, V.59, 1950 (`http://www.loebner.net/Prizef/TuringArticle.html`)

[2] In *The AI Magazine*, Fall 1982 (`http://www-rci.rutgers.edu/~cfs/472_html/Intro/MinskyArticle/MM1.html`)

[3] In *The Blind Watchmaker*

Several decades later, AI research has still a long way to pave until the real machine intelligence (or AI) as envisioned by the pioneers of the field is achieved. Along the path some researchers saw the potential of using nature-inspired algorithms to solve design, learning, and optimisation problems. The field of Evolutionary Computation (EC) [Eiben and Smith, 2003] is one example of this scientific endeavour, and is now mature. Its beginnings can be traced back to the 1960's. At that time, inspired in Darwin's *theory of natural selection*, different researchers started developing techniques to evolve artificial systems. The first three areas of study arising were evolutionary programming (EP), genetic algorithms (GA), and evolution strategies (ES). Only by the early 1990's the separate areas were unified as different representatives of the same technology - EC. At the same time, another stream following the same principles emerged: Genetic Programming (GP) [Koza, 1992]. GP is the result of an attempt to deal with the automatic programming of computers, a specialisation of the central question of AI presented before, posed by Arthur Samuel in 1959: how can computers learn to solve problems without being explicitly programmed? Or, in other words, how can computers be made to do what needs to be done, without being told exactly how to do it?

These approaches are reminiscent of the Central Dogma of Biology [Hartl, 2014] that posits an unidirectional relationship among DNA, RNA and Proteins. In fact, evolutionary algorithms are a simplification of this idea, for they implement a simple and direct (one-to-one) mapping between the genotype and the phenotype. Even if there are different algorithms each of them tuned for a specific problem or class of problems, they all rely in a similar approach: (1) randomly define an initial population of solution candidates; (2) select, according to the fitness, some individuals for reproduction with variation; (3) define the survivors for the next generation; (4) repeat steps (2) and (3) until some condition is fulfilled. Typically, the objects manipulated by the algorithms are represented at two different levels. At a low-level, the genotype, the representations are manipulated by the variation operators; at a high-level, the phenotype, the objects are evaluated to determine their fitness and are selected accordingly. Because of that, we need a mapping between these two levels.

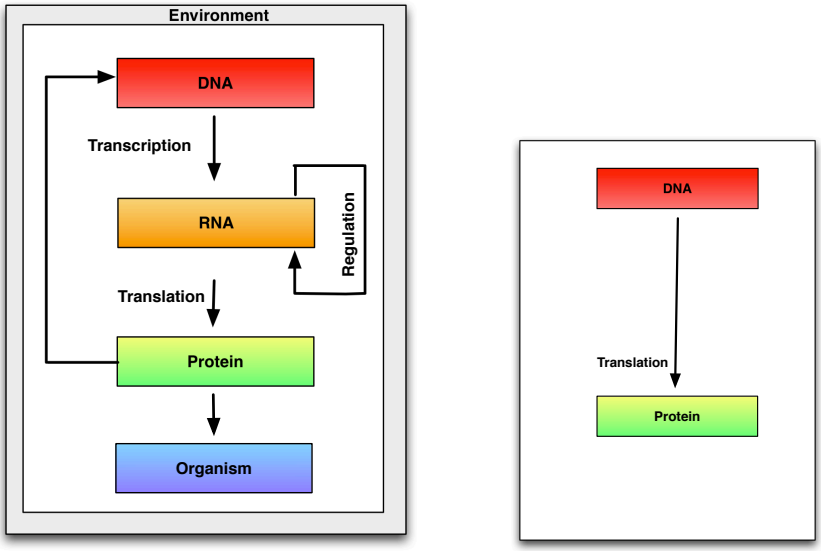
Biological knowledge has greatly evolved since the times of the modern synthesis of evolution. Today it is clear that there are many processes and mechanisms that influence

the way we came to being, involving evolution and development [Carroll, 2006, Davidson, 2010, Jablonka and Lamb, 2005]. The study of how the environment can influence evolution and development led to the appearance of computational epigenetics, aiming to understand the mechanisms of inheritable gene regulation. Thus it is indisputable that there is much more to evolution than just natural selection.

Two particular advances marked the last years. The first was finding that the genes governing the construction of major parts of organisms' bodies have exact counterparts in most animals, humans included. The second was the discovery that the development of several body parts which are very different across species (like eyes, fingertips, and others), is also governed by the same genes in different animals. Moreover, remarkable similarities in the genetic codes of species have been found thanks to the development of genome sequencing technologies, showing that the evolution of species is achieved through small changes in existing genetic material, rather than built from scratch. The central point of these findings is that gene expression is governed by different switches that when activated/deactivated in different times of the developmental process, and in different environments yield different outcomes. These switches form complex dynamic gene regulatory networks (GRN) whose outcome varies throughout the developmental process. Moreover, small changes in the DNA (the blueprint) of the GRN can yield different timings of gene expression, which in turn may produce a significant phenotype modification.

Many of these aspects have been studied computationally. There are studies about the role of DNA methylation in the context of Artificial Life Models [Sousa and Costa, 2011] or about the importance of histone modifications in the realm of Genetic Programming [Tanev and Yuta, 2008]. Gene Regulatory Networks (GRN) have been intensively studied in recent years, from a biological perspective, to understand their structure and dynamics [Davidson, 2010]. These studies are supported by computational simulations [Schlitt and Brazma, 2007, Geard and Willadsen, 2009]. These biological and computational models of GRNs have made clear the role of regulation in development [Carroll, 2006, Shubin, 2009]. Moreover, many computer scientists are well aware of the potential of applying the new ideas of development to problem solving in computer systems [Kumar and Bentley, 2003].

One of the criticisms addressed to the traditional evolutionary algorithms mentioned before, is that they are a simplification of nature's principles, for they implement a simple and direct (one-to-one) mapping between the genotype and the phenotype. There is nothing in between, and there are no regulation links. For example, typically in an EA, the two phases of transcription and translation are merged into just one and the regulatory processes are missing. At a larger scale, we could add the lack of epigenetic phenomena that contribute to the evolution and all the mechanisms involved in the construction of an organism (see Figure 1.1).



(a) The natural way (b) The artificial model

Figure 1.1: From genes to an organism.

It is thus without surprise that recently some researchers, reflecting about the new insights and comprehension of biological systems, proposed a research agenda whose main goal is to foster the inclusion, at least partially, of this knowledge into Evolutionary Algorithms, transforming the artificial evolution perspective into a computational evolution approach [Banzhaf et al., 2006]. They criticise many of the assumptions made by artificial evolution, namely: (1) the genotype is composed of discrete and independent

units, (2) the absence on non-coding material, (3) the environment doesn't influence neither evolution nor the development, (4) the genetic material is transferred directly to offsprings, (5) variation operates only at the genotype, (6) selection operates only at the phenotype level, and (7) the organisation of the genotype does not change during evolution. All these limitations can be summarised in just three aspects: (a) the influence of the environment, (b) gene expression, and (c) development. One can abstract further these three aspects to end up with just one concept: gene regulation.

The working hypothesis behind the automatic computation methodology described in this thesis is that EC representations can be improved by integrating these different perspectives of the same mechanism (regulation) in the architecture of evolutionary algorithms, with the aim of solving complex design, learning, and optimisation problems.

The main contribution of this thesis is a new representation for GP based on GRNs, and aimed at improving the results of traditional EAs. For that purpose a model of artificial GRNs was used as the genotypic representation, coupled with an algorithm to reduce the complexity of the network graph and map the artificial GRN into a program. Asexual variation operators were also developed, as well as recombination operators, to deal with the specific genotypic representation. Variants to the method are also described, extending its capabilities to a broader range of problem classes, from the traditional GP benchmarks to domains with intrinsic recursion, and problems that require solutions with multiple outputs. Moreover, a modular evolutionary framework was developed[4] (using Python), that allows the usage of different representations both at the genotypic and phenotypic levels.

The experimental work that was carried out studied some design alternatives of the underlying artificial GRN model (aimed at a better tuning of the algorithm). Experiments were also designed to assess the efficiency of the variation operators with different parameterisations. The best parameterisation found in these experiments was used as the basic configuration to handle the remaining classes of problems.

[4] Publicly available under the GNU General Public License 3.0+, at `https://github.com/rmlopes/code`.

The remaining of this thesis is organised as follows. The biological background, and the state-of-the-art on GRN models and EAs are presented in Chapter 2. Then the new algorithm and its feedback variant are described, along with the variation operators in Chapter 3, followed by the benchmark problems used to assess the algorithm's performance, in Chapter 4. The setup for the different experiments that were carried out is then presented in Chapter 5, and the corresponding results are presented and discussed in Chapter 6. Following this, an extension of the model which deals with multiple outputs is also studied in Chapter 7. Finally, the author's remarks and possible future endeavours are presented in Chapter 8.

2 State of the Art

The origin of life and of the Universe are the two fundamental problematics of both religion and science [Dawkins, 2008]. While in religion there is no knowledge *per se*, what we learn from science is meaningful, even though provisional. As Nick Lane wrote: "It is a joy to be alive at this time, when we know so much, and yet can look forward to so much more" [Lane, 2010].

In this chapter, the main characters and their contributions to our current body of knowledge are briefly presented in Section 2.1 [Jablonka and Lamb, 2005, Carroll, 2007]. Particular emphasis is given to gene regulation and expression in Section 2.2 [Carroll, 2006, Nüsslein-Volhard, 2006], which is the main biological mechanism behind the hypothesis of this thesis. An overview of the use of analogies and models that incorporate this biological knowledge into Evolutionary Algorithms is then presented in Section 2.3. The proposals described represent successful approaches to some of the issues mentioned in the previous chapter, and its results will be used as reference for the discussion in Chapter 6.

2.1 Background

What is life? How did it start? How did man come into existence? These are questions that have troubled mankind since before the ancient greek society. Common thoughts amongst greek philosophers were that life had spontaneously generated from water or air, and that there was a common guiding principle from which all living things were created. In particular, Aristoteles studied the development of chicks from the egg and pro-

posed a theory for the creation of living beings based on the (not well-known) mechanisms of reproduction.

This view of spontaneous generation and reproduction with the passing of information between generations was left aside during the medieval era, because of the Christian theory of special creation: all living beings came into existence in their current form, designed by divine intervention. Although some philosophers proposed theories closer to evolutionary thought, these were still relying on a divine form as the starting point of life.

It was not until the late 18th century that a theory of evolution was proposed. The revolution in evolutionary thinking started with George Leclerc - Comte de Buffon, and Erasmus Darwin. The first argued that living beings do change over time, probably as the result of environmental influences or even chance (although he still believed that the original form would have been spontaneously generated). The latter wrote poetry by which he not only claimed that all warm blooded animals had a common ancestor, but also had some vague ideas of what might be responsible for the changes in species. He was the first to discuss the effects of sexual reproduction and competition on possible changes in species. However, a mechanism that drives modifications from generation to generation was still missing.

Jean Baptiste Lamarck, one of Buffon's pupils, was the first to theorise a mechanism for heredity, and thus a theory of evolution. The main argument of his theory was that organisms are not fixed and they must adapt in reaction to environmental changes in order to survive, passing those changes along to the descendants. The most known example is perhaps that of the giraffe's neck. According to Lamarck as the giraffes stretched their necks to reach leafs their necks would become longer, this trait would be inherited by the offspring, and over several generations the continued stretching would result in the longer necks. Accordingly, organs that are not used by the organism would tend to shrink. Besides the inheritance of acquired traits, he also argued that organisms were continuously evolving from simpler to more complex forms, towards perfection. Moreover, species would not become extinct but rather change into other species.

Lamarckian evolution was the first public statement of a theory for the evolution of life as the result of natural processes rather than divine intervention. In the highly clerical society of the 18th century these revolutionary ideas were not well accepted. Moreover, his theory was easily discredited since, for instance, a cowboy's offspring is not born with

arched legs. Despite him being discredited and exiled from the scientific community, the notion of evolutionary change did not die with Lamarck[1].

In the 19th century, following the ideals of his grandfather, Charles Darwin also delved into the study of the diversification of species. Because of the society's reluctance in accepting revolutionary ideas, he worked quietly for decades on his theory, collecting as much observational data as he could, anticipating possible refutations. His work focused mainly on breeding experiments with pigeons and his thorough survey on the species of South America, on board of the *Beagle*. Alfred Russel Wallace, another great naturalist of the time, was also intrigued by the diversity of life. In 1855 he published "On the law which has regulated the introduction of new species", which showed that he was reaching similar conclusions to Darwin. After exchange of thoughts and observations (by correspondence), Wallace shared his essay "On the Tendency of Varieties to Depart Indefinitely From the Original Type", which would be presented in 1858 jointly with excerpts from an unpublished essay previously written by Darwin.

Following this, in 1859 Darwin publishes "On the Origin of Species", where he exposes his theory of natural selection as the driving force of evolution. Darwin's theory states that variations in the traits of species are essentially random, and that the most adapted individuals (with better traits) are more likely to survive and reproduce, thus passing the characteristic traits to the offspring (later coined as the *survival of the fittest*). Despite failing to explain the reproduction heritability mechanisms or the origin of new traits, natural selection was the first plausible theory for the evolution of species from a common ancestor.

In 1865 Gregor Mendel, an Augustinian friar and botanist presented his work with pea plants, demonstrating that the inheritance of certain traits followed particular patterns. He formulated two generalisations, later coined as the "Mendelian laws of inheritance". The first states that an individual has two factors for each trait, one from each parent (diploidy). The different forms of a factor are called alleles. During reproduction the offspring receives a random allele from each parent, and the genotype of an individual is the complete set of alleles. The second states that separate genes for different traits are passed to the offsprings independently from one another. Despite the paramount

[1] As a matter of fact, the inheritance of acquired traits is still discussed in the 21st century, in the context of epigenetics. The concept of epigenetics is not discussed here, but the interested reader can refer to [Jablonka and Lamb, 2005] for an introduction.

importance of his work it would only be recognised some decades later when it was rediscovered by other researchers.

At the end of the 1880s August Weissmann, a german evolutionary biologist, made an important contribution with the germ plasm theory. According to this theory in eukaryotic organisms inheritance takes place only through germ cells (egg cells and sperm cells). The somatic cells (the other cells of the body) do not work as agents of heredity. In other words, genetic information does not pass from soma to germ plasm, and into the next generation. One can easily see how this view refutes Lamarck's theory of acquired characteristics.

By the end of the century Hugo deVries was working on inheritance and reached approximately the same observations as Mendel. He concluded that the inheritance of specific traits in organisms comes in particles (the Mendelian elements), which he called *pangenes* (decades later shortened to *genes*). DeVries' work led to the rediscovery of Mendel's work, who is nowadays considered the "father of genetics".

DeVries became best known for his *mutation theory*. He coined the term mutations for the variations found in successive generations of a plant. He postulated in his theory that evolution, especially the origin of species, might occur more frequently with large-scale changes (the ones he called mutations) rather than via the continuous Darwinian evolution. Moreover, he was the first to suggest the occurrence of recombinations between homologous chromosomes, which is nowadays known as *chromosomal crossover*.

At the beginning of the 20^{th} century there were still many reluctant to accept that evolution had happen through natural selection. Moreover, many theories were thought to be contradictory, like the Mendelian concepts of hard-inheritance, or the large-scale changes observed by palaeontologists, against Darwin's natural selection and its continuous view of evolution. Also against Mendelian genetics were biometricians, for whom empirical evidence indicated that variation was continuous in most organisms.

Ronald A. Fisher was the first to reconcile some of these theories. He first showed how the continuous variation measured by the biometricians, could result from the action of many discrete genetic loci. His work culminated with the publication of his book "The Genetical Theory of Natural Selection", and he was able to show that Mendelian genetics is actually compatible with the notion of evolution driven by natural selection. John B. S. Haldane corroborated his work by applying mathematical analysis to real world examples of natural selection. Sewall Wright in turn focused on gene complexes, that is, combi-

nations of genes that interact with each other. This was the beginning of the discipline of population genetics. Fisher and Wright were also the first to propose a mathematical model for the random genetic drift (alleles frequency) in populations (rather than only change guided through natural selection), suggesting it plays a minor role in evolution and showing that its effect varies with the population size.

A unifying theory of evolution was still missing though. Most naturalists believed that there was more to evolution than natural selection could explain. It was the work of Theodosius Dobzhansky "Genetics and the Origin of Species" that brought together geneticists and naturalists. It presented the same conclusions as Fisher, Haldane, and Wright, in a form that was more accessible to others. Moreover, it emphasised the importance of genetically distinct sub-populations, and that the real-world populations had much more genetic variability than what was thought. This work was complemented by Edmund Ford, who was the first to define *genetic polymorphism*, that is, the maintenance of different phenotypes within the same population of a given species. In 1942, Ernst Mayr emphasised the role of allopatric speciation as a mechanism for the emergence of new species, where geographically isolated sub-populations diverge so far that reproductive isolation occurs. Moreover he introduced the biological species concept, defining a species as a group of potentially interbreeding populations that are reproductively isolated from all other populations. In the same year, Julian Huxley published "Evolution: the Modern Synthesis". In his book he uses the term "evolutionary synthesis" to refer to general acceptance of two conclusions: i) continuous evolution could be explained in terms of small discrete genetic changes and recombination (micro-evolution); ii) speciation (macro-evolution) could be explained consistently with the known genetic mechanisms. In 1944, George Gaylord Simpson showed that under careful examination, the fossil records of mammals were consistent with the mechanisms of population genetics known at the time. He argued that the gaps found in the fossil records which are inconsistent with the irregular branching, and non-directional pattern predicted by the modern synthesis could be explained by rapid *quantum evolution* in small populations. Quantum evolution suggested drastic shifts in small populations, where transitional forms would be unstable, and perish quickly therefore providing little fossil evidence. He complemented his theory with the *random genetic drift* proposed before by Wright as an adaptive mechanism within species, arguing that only rarely it could have driven a transition to a new adaptive zone (a new species). Despite controversy, his work was crucial since at that

time most palaeontologists did not believe that natural selection was the main evolutionary mechanism. Finally, the synthesis was extended by George Stebbins to encompass botany, including the important effects of hybridisation and polyploidy (multiple homologous chromosomes) in some species of plants.

Despite the synthesis being widely accepted, there were still many open questions in the entire evolutionary process. For instance, the referenced assorted elements (genes) that are transferred to the offsprings were still an abstract entity. Chromosomal recombination was accepted, the nucleic acids were known, but the role of DNA in heredity was not yet confirmed. This happened in 1952 and one year later James Watson and Francis Crick proposed the double-helix model of DNA structure, providing a physical basis for heredity. Not long after Crick stated the *Central Dogma of Biology* in 1958, explaining the flow of genetic information within a biological system: $DNA \rightarrow RNA \rightarrow protein$. It states that genetic information cannot be transferred back from protein to either nucleic acid. However, how exactly genetic information produced proteins was still to be unveiled.

In the beginning of the 1960's François Jacob and Jacques Monod explained for the first time how gene expression is controlled, proposing the first model of gene regulation in prokaryotes (this model is described with more detail in Section 2.2). RNA was known to have the information necessary for the ribosomes to produce the chain of amino-acids we call protein. How the information was delivered to the ribosomes remained a mystery, until 1961 when Jacob, Monod, Sydney Brenner, and Matt Meselson discovered the existence of another molecule (which they called messenger RNA, mRNA) that presents to the ribosome the information contained in DNA. Following this work, in 1966 the genetic code was "cracked", that is, it was deciphered which codons code for which amino-acid. This was the point of departure to read the human genetic code, and genome sequencing methods started being developed.

In the light of the growing knowledge in molecular biology, two refinements of the Darwinian evolutionary theory emerged in the early 1970s. First, the gene-centric view of Richard Dawkins. He metaphorically described genes as "selfish", since he theorised them to be the basic units of evolution, the organism serving only the purpose of hosting so that successful genes can replicate. In his own words [Dawkins, 2006]: "They are the replicators and we are their survival machines. When we have served our purpose we are cast aside". This was controversial, applauded for providing an extended explanation to natural selection, but also highly contested for the oversimplification of the relation-

ship between the genes and the organism. Second, in 1972 Stephen J. Gould and Niles Eldredge proposed the *punctuated equilibrium* theory. The idea behind this extension to the evolutionary theory is that species do not slowly change or morph from one to another (continuous evolution as first proposed by Darwin, also referred to as *gradualism*), but rather they experience long periods of stagnation (morphological stasis), and major evolutionary changes that lead to speciation are rapid and rare. Moreover, they oppose to Dawkins "selfish gene", advocating that genes act more as a record of evolutionary change, attributing the main role to the organisms since these are the entities interacting with the environment.

Until 1977 mRNA was thought to be a faithful copy of DNA, which is true in bacteria. However, Richard Roberts' and Phil Sharp's labs showed that the genes in eukaryotic animals contain many interruptions, named introns. These introns are cut out from the mRNA before it is translated into protein by the ribosome. The discovery of these split genes modified the way researchers thought about the architecture of the genome, so far based only on experimentation with bacteria. Typically the introns contain 90% of the DNA sequence in the whole gene, and were thought of as *junk-DNA* (sequences that apparently serve no purpose, although its function may just be unknown).

In the same year, Gould published a book that explored the relationship between embryonic development (ontogeny) and biological evolution (phylogeny). He showed how variations in the timing and rate of development (heterochrony) can provide the raw material upon which natural selection can operate, and its influence in macroevolution (the major evolutionary transitions). Entitled "Ontogeny and Phylogeny" his work was to a large degree the inspiration behind the modern field of evolutionary-developmental biology (*evo-devo* for short). It is worth mentioning that in 1917, D'Arcy Thompson postulated that differential growth rates could produce variations in form. In his book "On Growth and Form", he showed the underlying similarities in body plans and how geometric transformations could be used to explain the variations. This was not taken into account in the modern evolutionary synthesis, perhaps because Thompson advocated his theory as an alternative to the survival of the fittest.

Before the end of the 20^{th} century the biological world would be revolutionised yet again by the discovery of mutated versions of the homeotic genes in different species [Nüsslein-Volhard, 2006]. These genes act as switches for other genes and can themselves be induced by other transcription factors (proteins that inhibit/enhance the tran-

scription of a gene) and/or morphogens (signalling proteins). First identified in fruit flies by Edward B. Lewis in the 1930's, this set of genes controls the embryonic development of the organism, being related to the segmentation of the body, and the controlling the construction of the different body parts. The identification of the same homeotic gene complexes (with mutations) across different animals provides further proof for the theory of the common descent. Despite the high conservation of these genes, the resulting differences in gene regulation can be highly divergent. The conclusion to be drawn here is that the variety observed at the level the phenotype does not arise so much from individual variations in the genes, but rather from the second-order effects of these variations on the expression and timing of the gene networks.

2.2 Gene Regulation

The study of gene complexes and their interactions dates to the time of the modern evolutionary synthesis. But it was not until 1960 that with the available knowledge from studying populations of *Escherichia-coli* F. Jacob and J. Monod made the distinction between structural and regulatory genes, and described the first well-understood example of how gene expression is controlled. It is still used nowadays to explain gene expression and regulation as it is a simple example.

A gene is expressed when the DNA is transcribed and translated to produce proteins (as stated by the central dogma of Biology). The central dogma explains the basic process of gene expression into proteins, but is unable to explain several essential phenomena such as cellular differentiation, where cells with the same genetic information behave differently according to their function in the organism [Davidson, 2010].

In bacteria (prokaryotes) a set of genes with related functionality and a common control of gene expression is called an *operon*. Jacob and Monod studied the *lac-operon*, the set of genes controlling the expression of enzymes that metabolise lactose, one of the possible sources of energy. This operon only needs to operate if lactose is present and glucose is not, as this is the preferred energy source. Figure 2.1 shows a lactose molecule. The function of the enzymes produced by the lac-operon is to breakdown the lactose molecules.

In Figure 2.2 one can see a summary of the activities involved in the expression of the enzymes responsible for metabolising lactose molecules. Three genes are transcribed

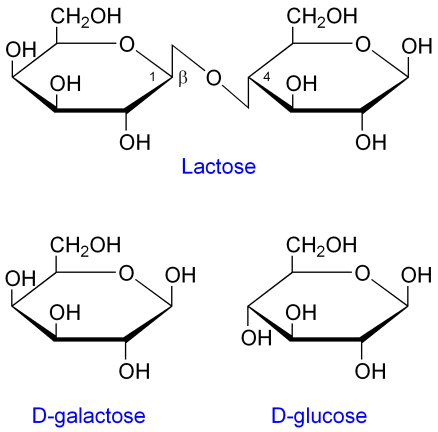


Figure 2.1: The lactose molecule and the break-down products (from WikimediaCommons, by Telliott, Public Domain).

into RNA and each codes for one of the enzymes. There are two regulatory regions, the promoter and the operator. The former enables the start of the transcription, and it is where the RNA polymerase binds to begin transcribing, as long as the latter is free. As one can see in Figure 2.3 there is a repressor protein (lac-repressor) that binds either to the operator or to lactose molecules. If lactose is present it will bind to the repressor, changing its structure, and unbinding from the operator. Otherwise it blocks the RNA-polymerase from transcribing the genes. This is the default behaviour and is known as *negative control*.

But there is also a *positive control* mechanism in this example. The main actor of this mechanism is the CAP (catabolite activator protein) which can bind to the DNA if glucose levels are low, and enhances the binding of the RNA-polymerase, thus increasing the final enzyme levels. If glucose levels are high, the CAP production is inhibited and although the genes are still expressed in the presence of lactose, the expression levels will be low.

This provides the basic example of gene expression and regulation in prokaryotic organisms. For further details the reader can refer to [Jablonka and Lamb, 2005, Carroll, 2006]. In eukaryotes the regulatory mechanisms are more complicated as for instance there are no operons, and multiple regulatory regions can be found for different transcription factors. Nevertheless the principles of gene regulation and expression are similar.

[2]The product of the lac Z gene is β-galactosidase, which converts lactose into glucose and galactose; the lac Y gene produces β-galactoside permease, that transports lactose into the cell; finally, the lac A gene is responsible for the production of β-galactoside transacetylase, whose function as part of the lac-operon is unknown.

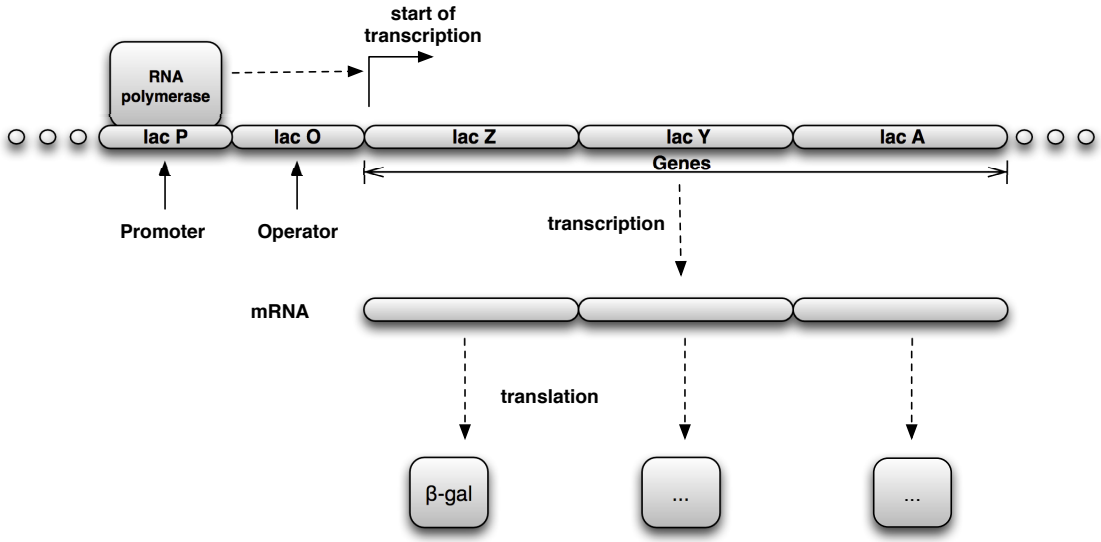


Figure 2.2: Transcription and translation in prokaryotes, particularly the lac-operon as presented by Jacob e Monod. The RNA-polymerase binds to the promoter region and, if the operator site is free, transcribes the three contiguous genes into mRNA. The resulting strand is then translated into the enzymes that will metabolise lactose[2].

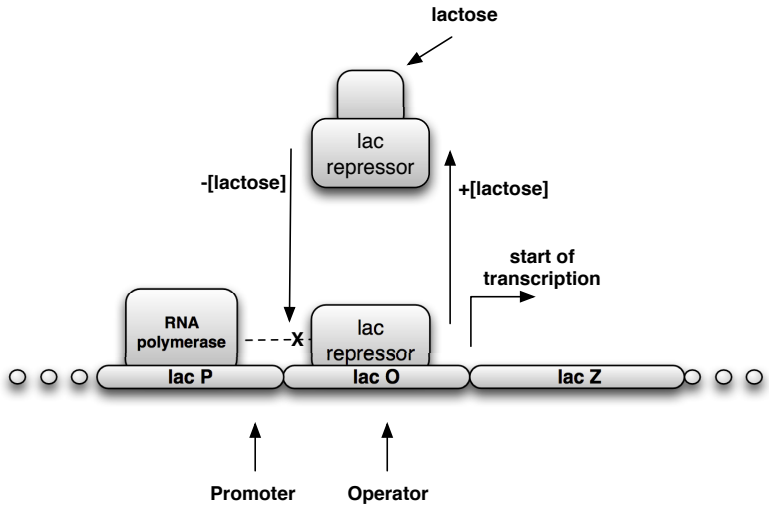


Figure 2.3: Negative control in the lac-operon as presented by Jacob e Monod. The lac-repressor inhibits gene expression by binding to the operator. When the concentration of lactose increases it binds to the repressor releasing the operator and enabling transcription. When lactose levels fall, the repressor is free and binds again to the operator.

In eukaryotic organisms, the gene regulation mechanisms result in differential gene expression, and this leads to cell specialisation. It is known since Watson and Crick that the eukaryotic DNA is tightly wrapped by histones, which provide one of the mechanisms for interaction between proteins and genetic material (what is called epigenetic control). In the next paragraphs the focus will be on the regulatory mechanisms rather than the epigenetic ones, since it is more relevant for the theme of this thesis.

In terms of cellular machinery, the main distinction between prokaryotes and eukaryotes is that the latter have a well defined nucleus (amongst other structures) enclosed within membranes. Of course this leads to other major differences, particularly in cell division. There are two types of cell-division mechanisms. One is mitosis, by which a cell divides to produce two genetically identical cells. This is the procedure that builds a multi-cellular organism from a single cell during development. If the replicated cells have the same genetic information it must rely on differential gene expression to achieve differentiation. The other is meiosis, used in sexual reproduction by diploid cells (containing one chromosome from each parent). After two stages of cell division and recombination of each pair of parental chromosomes, it results in four haploid cells (known as gametes). Other important distinctions in eukaryotes include: i) three distinct types of RNA-polymerase; ii) most mRNA encodes a single product; iii) many genes contain introns, whose RNA product is spliced out before mRNA transport to the cytoplasm (leaving only exons in mature mRNA).

The control of gene expression can happen at different points of the protein production line, from the transcription stage (inside the nucleus) to the expressed product stage (Figure 2.4). These moment-to-moment adjustments are influenced not only by the environment, but also by internal and external signals, and may effect on both the quantities and structure of the produced proteins. Despite the many possibilities, the majority of gene control happens at the transcriptional level. Similarly to the bacterial mechanisms, different proteins (called transcription factors) can bind to the regulatory regions upstream or downstream of a gene, either helping or hindering RNA-polymerase from binding to the promoter region of the DNA in order to start transcription. Figure 2.5 presents a simplified model. For further information on the regulatory and signalling mechanisms in eukaryotes the interested reader can refer to [Bray, 2009].

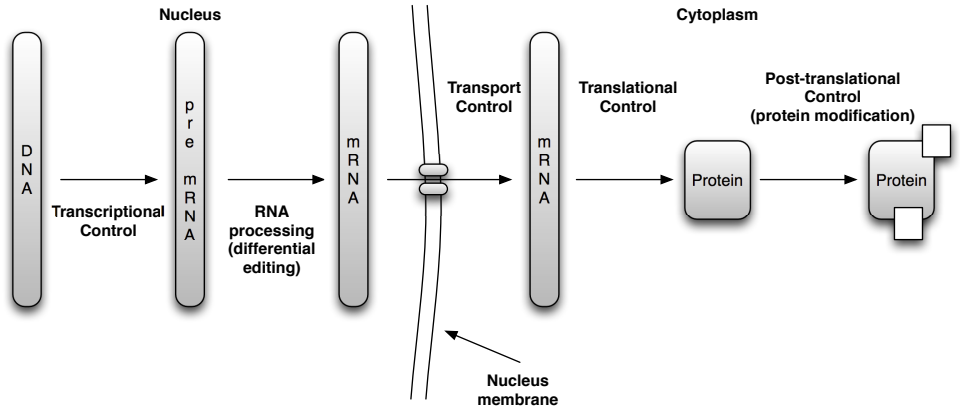


Figure 2.4: Gene expression in eukaryotes can happen at different points of the protein production line, from the transcription stage (inside the nucleus) to the expressed product stage.

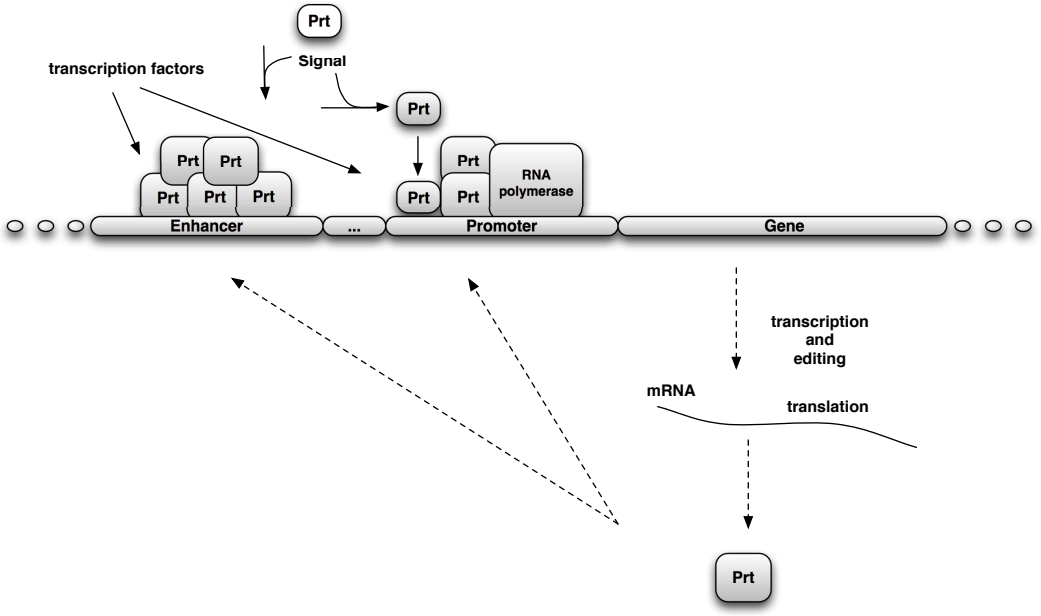


Figure 2.5: Simplified model of gene expression in eukaryotes. The transcription factors bind to the promoter and enhancer(s) sites providing varying levels of expression by either repressing or enhancing the transcription of the genes.

Since these regulatory networks are highly non-linear and have several thousand variables obtaining a model from biological data is a complex task. Computational approaches to this problem try to reconstruct the GRN from experimental data, for instance, from gene expression data provided by microarrays. Within this perspective devising a GRN is viewed as a reverse engineering problem, where you build a model from data. Attempts were made to use evolutionary algorithms to solve this problem for some of the formal models. In particular, there have been proposed solutions based on genetic algorithms [Marbach et al., 2007], genetic programming [Sakamoto and Iba, 2001], evolution strategies [Streichert et al., 2004], and differential evolution [Noman and Iba, 2007]. For surveys on the area the reader can refer to [Filkov, 2005, Choi et al., 2007, Hecker et al., 2009]

Various approaches for formally modelling gene regulatory networks (GRN) appeared in the last decades. The proposed models can be classified according to the following aspects: variables such as product concentrations are discrete, continuous or mixed; time is discrete and the update of the variables is either synchronous or asynchronous[3]; space is discrete, continuous or absent. Examples of models include directed graphs, bayesian networks, ordinary and partial differential equations, random boolean networks, neural networks, and rule-based formalisms. Reviews of the proposals can be found in [de Jong, 2002, Schlitt and Brazma, 2007, Geard and Willadsen, 2009, Vijesh, 2013].

2.3 Evolutionary Computation

Living beings are extraordinary creatures, with such complex mechanisms that only millions of years of evolution could have shaped. Witnessing the outstanding quality of the evolutionary result some computer scientists decided to mimic natural evolution in their computers, giving rise to a new biologically inspired research field - Evolutionary Computation (EC) [Eiben and Smith, 2003]. Over time several stochastic iterative population-based search algorithms (Evolutionary Algorithms - EA) were proposed, that have been especially tuned for some problems and/or situations (for instance, algorithms for dealing with noisy, uncertain, or dynamic environments, for evolving rather than designing the algorithm's parameters or some of its components, algorithms with local search operators,

[3] There are, however, cases where time is continuous

and for multi-objective optimisation).

The first approaches to EC emerged in parallel during the late 1960s: Evolutionary Programming (EP) [Fogel and Burgin, 1969] and Evolution Strategies(ES) [Schwefel, 1995] (originally published in german as an internal report, dated 1968). Both rely on the theory of natural selection actuating over the phenotypes, there is no distinction between genotype and phenotype, and the core procedure is about the same (Fig. 2.6): (1) randomly define an initial population of solution candidates; (2) select, according to the fitness, some individuals for reproduction (sexual or asexual) with variation; (3) define the survivors for the next generation; (4) repeat steps (2) and (3) until some termination condition is fulfilled.

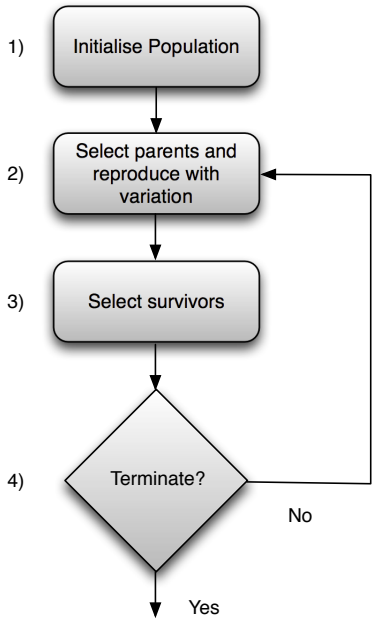


Figure 2.6: Flow diagram of an Evolutionary Algorithm.

The main difference between these two approaches is that while Lawrence Fogel addressed the problem of evolving artificial intelligence (that is, programs with predictive capabilities, specifically *finite state machines*), Schwefel and Rechenberg used vectors of real values to solve optimisation problems. Moreover, despite both using mutation as the main variation operator, EP focuses on behavioural changes in species, rather than behavioural changes at the level of the individual. Another difference is that selection

in EP is typically based on a tournament (solutions dispute a tournament based on the fitness value and the most ranked are selected) while in ES mainly greedy selection is used based directly on the fitness evaluation of each individual in the population.

Not long after a third approach was proposed by John Holland which stresses chromosomal operators: Genetic Algorithms (GA) [Holland, 1973]. The main difference of his proposal lies on the representation of the individuals. Solutions are now seen as genomes that codify the phenotypes to be evaluated. Moreover, it is based on Holland's schema theory which states that low-order schemata (or building-blocks) with improved fitness increase exponentially across the population in successive generations. Nevertheless the mapping from genotype to phenotype is direct, i.e., there is a one-to-one mapping from genotype to phenotype. Typically the values are represented as a binary string, and the variation operators actuate on the genotype rather than directly on the phenotype. The main variation operator in a GA is recombination, typically called crossover (mimicking sexual reproduction), in which two individuals contribute with genetic material to form two different offsprings.

In the early 1990's Genetic Programming emerged. John Koza's technique evolves computer programs (similarly to Fogel), inspired by the concepts of the GA's building-blocks. Programs are represented by trees, upon which the variation operators act, and are artificially selected according to their phenotypic effect (the program execution result). The main difference to EP is that the programs structure is evolved and the main operator is recombination. Differently from GA's, in GP the mapping from genotype to phenotype is still direct as the programs themselves are being manipulated by the operators. Nevertheless, it is well documented that introns (sub-trees with neutral effect) can be found in GP individuals [Langdon and Banzhaf, 2000] resulting in genotypes with the same phenotypic effect (the same execution result) and in the propagation of this code[4]. In the last two decades this paradigm has been applied with success in many different fields, from engineering to linguistics, and many others. Amongst the examples of such applications are antennae design [Lohn et al., 2008, Casula et al., 2009, Lohn et al., 2004], amplifier design [Koza et al., 1997b, Koza et al., 1997a, Koza et al., 1998], neural-network design [Esparcia-Alcazar and Sharman, 1997], and brain computer interfaces [Poli et al., 2011]. A compilation of *human-competitive* results by GP is presented in [Koza, 2010].

[4]This phenomenon of propagation of neutral code is called *bloat*; for a comprehensive study of bloat in GP please refer to [Silva and Costa, 2009]

By the end of the 1990's indirect mappings started emerging, introducing more complexity into the phenotype-genotype relationship by defining many-to-one mappings (different genotypes map the same phenotype). Whether this approach provides benefits and to what extent is a controversial question that remains. David Fogel argues that there is a misplaced emphasis on emulating the genetic mechanisms [Fogel, 2006]. Others have found self-adaptation capabilities deriving from these types of representation [Igel and Toussaint, 2003]. The point of this section is to provide an overview of the research made towards the inclusion of more realistic mechanisms into EC representations, rather than discuss which is better over which.

The different representations that have been developed as an alternative to the typical GP can be abstracted to two general categories. In the first, one has many-to-one, redundant mappings with linearised genomes which rely on recombination and mutation as the main variation operators. Although in some cases researchers argue to have used developmental analogies, there is no specific simulated development in these representations (Section2.3.1). The second is composed of proposals that integrate simulated development, usually more robust and reaching classes of problems that the more conventional methods would not allow (Section 2.3.2).

2.3.1 Non-developmental Representations

The proposals described in this section use many-to-one, redundant mappings, by which many genomes are mapped into the same phenotype. Typically the genome representation is linear (as suggested by the models of the natural genomes). Each offers advantages and disadvantages, but only a few were widely adopted by practitioners.

Grammatical Evolution

Grammar-Guided Genetic Programming (GGGP) is a class of population-based stochastic search techniques applied to context-free grammar-based representations[5]. A context-free grammar (CFG) is a tuple $G = (N, T, S, P)$, where N is a non-empty set of non terminal symbols, T is a non-empty set of terminal symbols, S is an element of N called axiom, and P is a set of production rules of the form $A ::= \alpha$, with $A \in N$ and $\alpha \in (N \cup T)^*$. N and T are

[5]Other types of grammars have been used, for instance Tree-Adjunct, or annotated grammars

disjoint. A language associated with a grammar G, $L(G)$, is the set of all sequences of terminal symbols that can be derived from the axiom, that is $L(G) = \{w : S \overset{*}{\Rightarrow} w, w \in T^*\}$.

Fitness evaluation is performed by reading an expression tree generated using the defined grammar, and evaluating it as a standard GP program. This method has the regular GP parameters (population size, maximum generations, maximum tree depth, and operator probabilities), as well as the same mechanisms for selection and reproduction. The crossover and mutation operators are slightly modified since now there is a grammar restricting the search space (see [Byrne et al., 2009] and [O'Neill et al., 2003] for a discussion related with this issue). Known examples of GGGP systems are the Context-free Grammar GP [Freeman, 1998], LOGENPRO [Wong and Leung, 1995], and Grammatical Evolution (GE) [O'Neill and Ryan, 2003].

GE is the most successful form of GGGP. It introduces a genotype to phenotype mapping, where the genome is a linear sequence which is translated into a derivation tree (the phenotype), and will be further decoded to an expression tree (the common approach approach to GGGP linearisation). Each codon[6] is usually defined as an 8-bit binary sequence (integers may be adopted instead), which is used to determine the rule for a non-terminal symbol when it is expanded. The biological analogy that is usually described includes development (non-simulated), viewed as the derivation process from the DNA (the binary string) until the proteins (the terminal operands), resulting in a program (the phenotypic effect), as shown in Figure 2.7.

Suppose that we have the following production rule,

$$
\begin{aligned}
&< expr >::= < expr >< op >< expr > && (0)\\
&\quad |(< expr >< op >< expr >) && (1)\\
&\quad | < pre-op > (< expr >) && (2)\\
&\quad | < var > && (3)
\end{aligned}
$$

where there are four options to rewrite its left hand side symbol $< expr >$. In the

[6] In biology, a codon is a set of three adjacent nucleotides in mRNA that specify the amino-acid to be produced; in GE it is a set of bits or an integer that define the grammar rule to be used in the next derivation step.

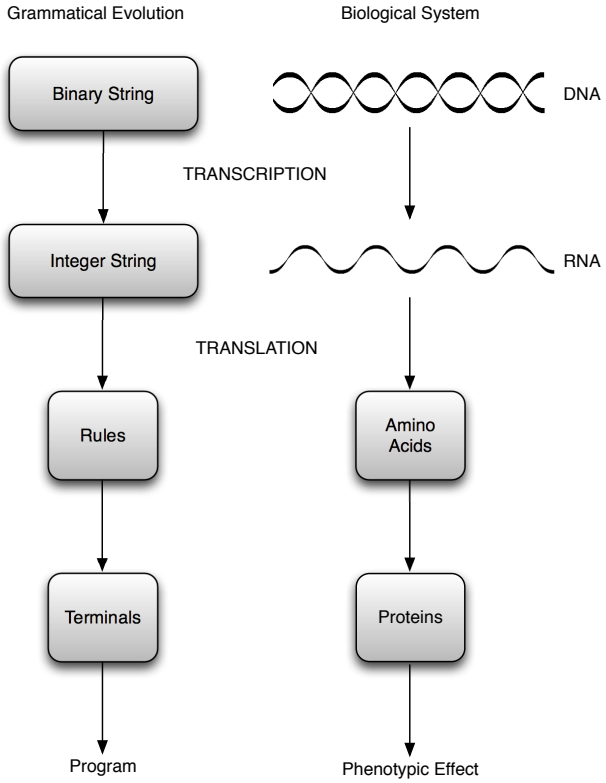


Figure 2.7: Grammatical Evolution analogy with biological systems, including non-simulated development, as the derivation from the DNA (the binary string) until the proteins (the terminal operands) resulting in a program (the phenotypic effect). Adapted from [O'Neill and Ryan, 2003].

beginning we have our genome transcribed into a string of integers and a syntactical form equal to the axiom $< expr >$. We want to rewrite the axiom and must choose which alternative will be used. We take the first integer and divide it by the number of options for $< expr >$. The remainder of that operation will indicate the option to be used. In the example above, if we admit that the integer is 9 then we will have $9\%4 = 1$ and the axiom will be rewritten in $(< expr >< op >< expr >)$. Then we read the second integer and apply the same method to the left most non-terminal of the derivation. We iterate this process, that stops when we don't have more non-terminals to rewrite. But what happens if we run out of integers, because we arrive to the end of the genome, and still have non-terminals to expand? This is when we use a wrapping mechanism: we restart from the beginning of the string of integers. The existence of redundancy is also worth noting, for different integers may correspond to the same alternative due to the nature

of the operation remainder. In the example above, the integers $5, 9, 13, \ldots$ all code the same production alternative for $< expr >$.

Three main innovations were introduced with this algorithm: genomes have variable length, redundancy, and wrapping is allowed. However, this linearised version still has some issues. First, there is positional dependence, that is, when a codon is moved to another part of the genome its function maybe very different. Second, an apparently valid genotype may code an infeasible phenotype (that cannot be evaluated). Finally, the locality principle (small changes is the genotype resulting in small changes in the phenotype) may not be fulfilled. As GE is the most widely used grammar-based system, many improvements have been proposed addressing these issues. For a comprehensive survey on grammar-based GP developments and applications please refer to [McKay et al., 2010].

Linear Genetic Programming

In Linear Genetic Programming (LGP) programs are written in an imperative language (like C), rather than in tree-based expressions, like in standard GP[7] [Brameier and Banzhaf, 2007].

In LGP an individual is a variable-length sequence of simple imperative language instructions. These instructions work with one, two, or three variables (the registers), or with constants from a predefined set, storing the result in a destination register (one of the registers is chosen as the output, and it is usually kept during the evolutionary process). The inputs to a given problem instance are given through the register's initialisation. Despite the linear representation, initialisation issues common in GP still apply: the user must set a lower and upper bound for initial population generation, influencing the performance of the search. The operators used are common to the standard technique, except that two mutations are performed: micro (which changes the properties of an instruction), and macro mutation (which inserts or deletes a random instruction).

For instance, 3-register instructions operate on two registers or constants, and assign the result to a third register. Such an instruction is represented by a vector of indices, as

[7] There is however similar work in the literature, using machine code [Nordin, 1994].

in the following example:

$$r_i := r_j + r_k$$
$$<id(+),\ i,\ j,\ k>$$

This representation allows to code an instruction as a 4-byte integer value, similarly to a representation as machine code, but it can be adapted to a particular processor. Thus, a program is represented by an array of integers.

This representation has the advantage of being able to directly run the generated programs without the need for an interpreter, thus making the evolutionary process quicker. Moreover, solving multiple-output problems is as simple as defining more than one output register. The main problem with this representation is the number of registers (or variables) to use. This parameter is problem dependent and increases in function of problem complexity, leading to poor results when inappropriately chosen.

Cartesian Genetic Programming (CGP)

CGP was first presented by [Miller, 1999, Miller and Thomson, 2000], and rapidly became a subject of interest for several researchers. In the original proposal the genomic representation is a grid of nodes, each represented by a string of integers, addressed in a Cartesian coordinate system. The integers define the nodes function, and how the connections between them are built. After identification of the output node, the solution is built backwards until the input nodes are reached (nodes without input connections), resulting in a feed-forward circuit. No crossover is applied, only mutation. Similarly to GE this representation introduces a genotype-to-phenotype mapping, illustrated in Fig. 2.8. In this case three different types of redundancy are introduced: node redundancy (genes associated with nodes that are not part of the connected graph); functional redundancy (a group of nodes implements functionality that could be implemented with fewer nodes); and input redundancy (when some node functions are not connected to some of the input nodes).

In [Miller and Thomson, 2000] CGP is applied to two different types of problems - regression and artificial ant - reporting very competitive results for the artificial ant. It was not compared in the symbolic regression case since the authors only used the fixed constant 1.0, in contrast to choosing random ephemeral constants, conferring great ad-

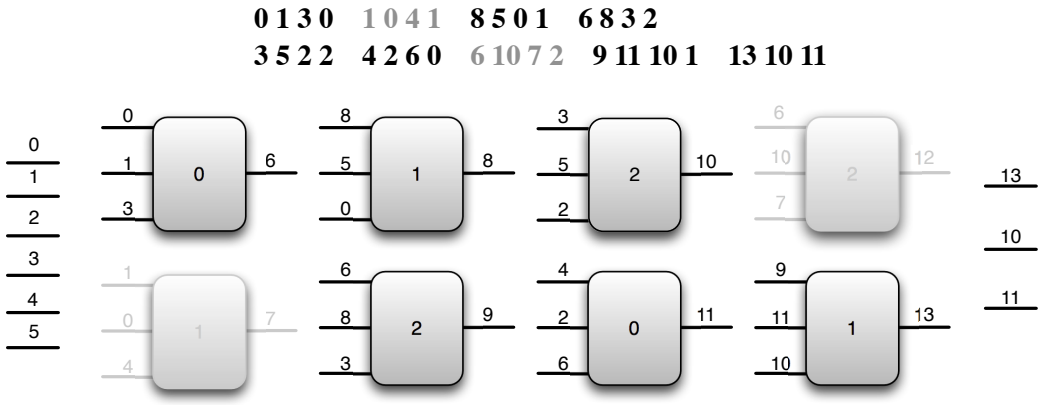


Figure 2.8: Adapted from [Miller and Thomson, 2000]: Genotype-phenotype mapping. Top: genotype. Bottom: phenotype. For a program with six inputs and 3 outputs, and three functions (0, 1, 2 inside square nodes, in italics in genotype). The grey squares indicate unconnected nodes.

vantage[8]. Despite the initial grid format proposal, several studies conducted afterwards have shown that the best representation is a linear genome, and this is now the common option for CGP. A notable extension is the Embedded-CGP (ECGP) [Walker and Miller, 2004a], that uses an operator to compress parts of the genome which are then used as sub-circuits, similarly to automatically defined functions. For a description of this representation and its extensions the reader should refer to [Miller, 2011].

Gene Expression Programming (GEP)

GEP is an alternative linear representation for GP that also introduces a new genotype-to-phenotype mapping. It was published by [Ferreira, 2002], and although less expressive than CGP it also revealed to be of interest for several researchers.

The genome in GEP is a linear fixed-length string composed of one or more genes with equal length. Despite the fixed length genes, the *open-reading frames* (ORF), i.e., the coding sequence of a gene, has variable length and thus it is common to have non-coding sequences at the tail of the genomes. As an example, consider the genome defined in Eq. 2.1. From this representation we can construct an expression tree[9], as illustrated in Fig. 2.9.

[8] There are well-known over-fitting issues in symbolic regression when using random constants

[9] The expression is read from left to right, constructing the expression tree in a breadth-first fashion.

```
012345678901 2345678901
*-/Qb+b+aaababaabbbbba                    (2.1)
```

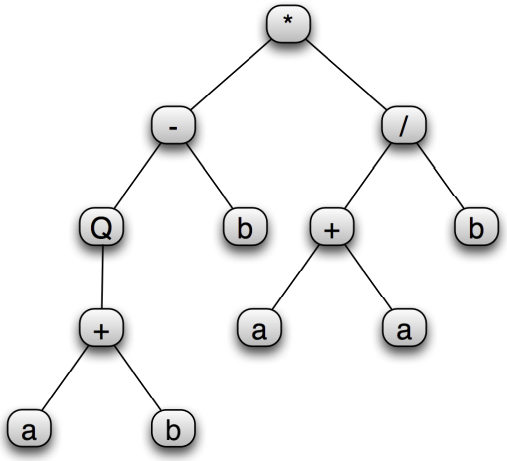


Figure 2.9: Reproduced from [Ferreira, 2002]: the expression tree for the gene presented in Eq. 2.1.

As one can see in the example, if there is more genetic material composing the gene than it is necessary, it will not be used. In [Ferreira, 2002] the author argues that "because of this apparently trivial fact, the genome of GEP individuals can be easily modified using any genetic operator. This means that, for the first time in evolutionary computation, a truly functional genotype/phenotype system is created in which the search space can be thoroughly explored by a myriad of genetic operators". This is more clear if one looks at the structure of the complete genes when the genomes are built.

Genes are structured in two parts: a head and a tail. The first may be composed of symbols from either the terminal or the function set, whether the second can only be composed of terminal symbols (so that ORFs will always terminate before the end of the gene). The head length (h) is problem dependent according to the author, whereas the tail length is a function of h and of the maximum arity. Using this scheme, no matter what mutation is performed, it will always render feasible expression trees, which may vary in length. Given this, multigenic chromosomes may be easily built by glueing genes to

one another with a randomly chosen function with appropriate arity. This promotes the evolution of complex modular hierarchical structures, where each small building block is coded by a gene.

The results reported in [Ferreira, 2002] show good success rates, although highly dependent on the right parameterisation of the number of genes, and the head and tail lengths. Moreover, like in GP, it revealed to be prone to the propagation of bloat (neutral sub-trees). Finally, the author also showed the importance of having neutral coding regions, by comparing multigenic less compact instances with unigenic compact ones.

Enzyme GP (EGP)

Enzyme GP [Lones and Tyrrell, 2001] is a form of GP that uses a biomimetic representation, which particularly mimics metabolic pathways between special proteins. The principle behind EGP is that the programs structure is not explicitly defined in the genome, but it rather results from the connection choices of the individual components (the enzymes) which carry a developmental process during evaluation resulting in a static executional structure.

The enzymes in an enzyme system have three attributes - shape, activity, and specificity - specifying respectively its structure (how it is seen by the other enzymes), its role in the system (wether it is an input, terminal, or some function), and the input preferences (similarly to the binding domains that determine which substrates will be bound, in biological enzymes). Consequently, input enzymes (or glands) do not have specificities (they do not receive inputs from within the system), and output enzymes (receptors) do not have a shape since their product is not used within the system.

The resulting executional structure is obtained from the interactions between enzymes, resembling the formation of metabolic pathways in biology. The receptors are the first to choose their substrates according to their specificities, until all the active enzymes have bound the necessary substrates. This is a deterministic process and thus a determined enzyme system will always develop into the same program. In [Lones and Tyrrell, 2002] the author provides a good illustration of this process, reproduced in Fig. 2.10.

In [Lones and Tyrrell, 2002], a slightly different implementation is provided, called the *functionality model*. This version uses a more advanced definition of shape which au-

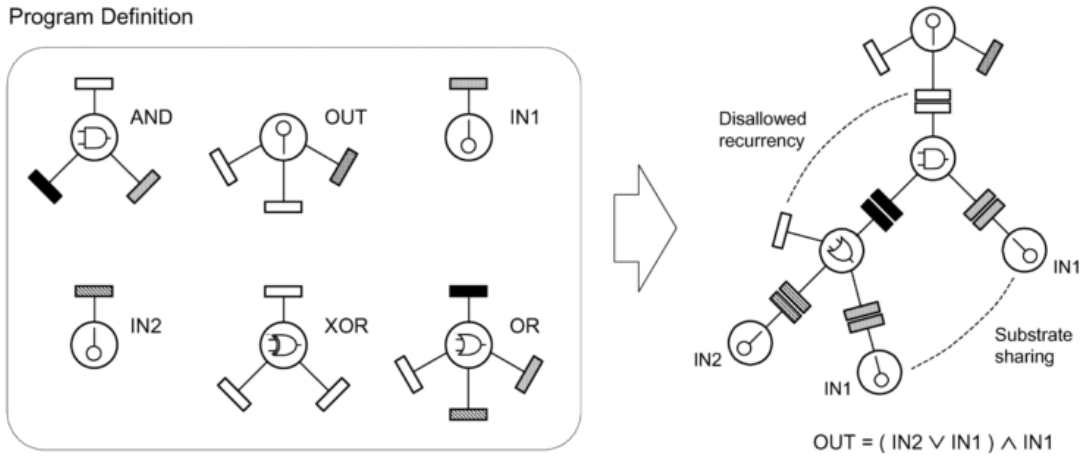


Figure 2.10: From [Lones and Tyrrell, 2002]: Development of an enzyme system. Shape is indicated by pattern and specificity strengths are not shown for clarity. Enzyme 'OUT' selects enzyme 'AND' as its input according to its strongest specificity. Enzyme 'AND' then selects its inputs, followed by enzyme 'OR'. However, due to the non- recurrence constraint, 'OR' may not choose 'AND' as an input. Enzyme 'IN1' is bound to more than one enzyme, demonstrating a capacity for reuse. Notice that enzyme 'XOR' is not expressed during execution.

tomatically supports variable-length programs, a limitation of the original *activity model*. The idea behind functionalities is that these are functions of the activity (role) of the enzyme and also its specificities, relating both the role and the preferences of the enzyme in one attribute. This is useful, for instance, when an enzyme does not have the appropriate substrate in the system. In the activity model when connecting to a less preferred substrate this does not have necessarily some relationship with the most preferred one, meaning that the enzyme's role could change dramatically in function of the environment. With functionalities it is possible to connect to a less preferred but similar substrate.

The evolution of the population is done in a distributed GA platform[10] but no comparative studies with a standard evolutive platform has been performed to this date. The genome is a linear collection of enzymes composed of a complete set of glands and receptors (common to every genotype), and enzymes. The operators used were a two-step crossover (combining gene recombination and shuffling, similarly to meiosis) and mutation.

[10]The individuals are distributed across a grid, and selection operates locally based on the neighbourhood of each grid position.

Its application in problems of logic design did not improve on known results from other techniques, although it showed some interesting properties, like the preclusion of bloat. Nevertheless, the results reported are very competitive with the ones presented in the literature.

2.3.2 Developmental Representations

The particularity of the proposals that will be presented in the remaining of this section is that they integrate some type of regulatory and/or developmental analogy into the algorithm, and have been used to solve typical optimisation and automatic programming benchmarks. Some of the models are extensions of those presented in the previous category. Several other developmental computational models have been proposed to investigate the intricate relationship between evolution, heredity, and development. The interested reader can find examples of these models in [Kumar and Bentley, 2003], as well as a review of artificial development results in [Harding and Banzhaf, 2008].

Ontogenetic Programming

[Spector and Stoffel, 1996] proposed an enhancement to GP, inspired by Koza's automatically defined functions (ADFs), and as an alternative dynamic version of these[11]. The authors implemented functions that are able to modify the program during run-time, using their technique in a Lisp implementation of HiGP, a stack-based GP framework.

The analogy to development was provided by three self-modifying functions: *segment-copy*, *shift-left*, and *shift-right*. The first replaces a part of the program with a segment copied from another location of the genome. The second and third functions rotate the code of the program by some number of instructions, either to the left or to the right respectively.

The model was applied to the prediction of a binary sequence and compared against regular HiGP, showing that it would be impossible to solve without the self-modification functions. The methodology was not widely used by other researchers, but as one will see in the next section it became a model of reference in the field.

[11] ADFs are pre-evolved functions which are used as part of the function set, and are not changed during run-time

Self-Modifying CGP

Self-Modifying CGP was first published by [Harding et al., 2007], and it was revised in 2009 [Harding et al., 2009b, Harding et al., 2009a, Harding et al., 2010], solving some issues of the first version. As the nomenclature indicates this is an extension of CGP (see Sect. 2.3.1) that allows a phenotype to modify itself during evaluation (the developmental process). The technique is essentially the same as described in the "Ontogenetic Programming" model, although in this case it is applied to the CGP representation.

The inclusion of self-modifying functions is the main difference to the original version of CGP. These allow the phenotype to manipulate its own structure by duplicating a node and its surrounding neighbours, or by deleting nodes, amongst other operations, in consecutive iterations of the circuit. As an example, the duplication function is presented in Fig. 2.11, reproduced from [Harding et al., 2009a]. Another important change was on the input function, which at each iteration addresses the next input in the inputs-set, instead of addressing a fixed one.

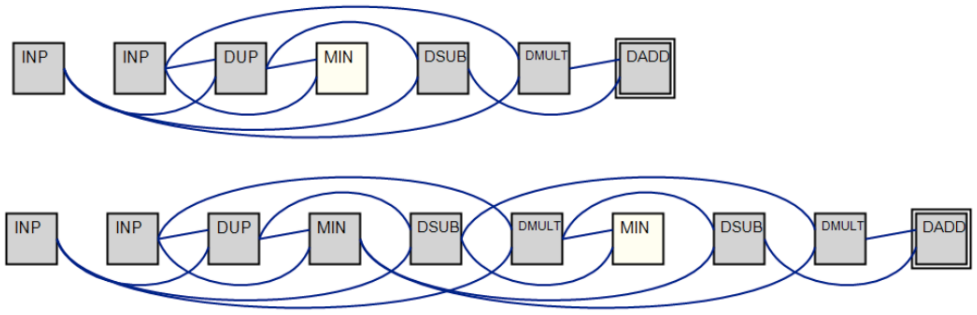


Figure 2.11: In [Harding et al., 2009a]: showing the DUP(licate) operator being activated, and inserting a copy of a section of the graph (itself and a neighbouring functions on either side) elsewhere in the graph in next iteration.

These modifications allowed the authors to solve a different class of problems, where a static structure will simply not be able to reproduce the desired behaviour. Examples of this class of problems are the n-bit parity generator, the Fibonacci sequence, the squares sequence, and learning tasks, amongst others. For a comprehensive survey on SM-CGP techniques and applications please refer to [Harding et al., 2010].

Fractal Proteins

In 2009, Krohn et al. presented a model to compute the digits of π [Krohn et al., 2009], based on the interactions between proteins (similarly to Enzyme GP, see Sect. 2.3.1). This work was motivated by and relied on an evolutionary model of fractal proteins, described in [Bentley, 2004]. Bentley demonstrated that it was possible to evolve regulatory genes that code for complex fractal proteins (subsets of the Mandelbrot set), which then interact forming a dynamic GRN that can exhibit interesting behaviours.

The representation for generating the digits of π using fractal proteins comprises different types of objects, illustrated in Figure 2.12. As mentioned before, the fractal proteins are represented as subsets of the Mandelbrot set; the *Environment* is composed of one or more proteins (expressed from the environment gene(s)) and one or more cells; the *Cell* comprises a *genome* and *cytoplasm*, and has some behaviours; the *Cytoplasm* contains one or more fractal proteins; the *Genome* is formed by *structural genes* and *regulatory genes*; a *regulatory gene* is composed of a promoter and a coding region; a *cell receptor gene* is a *structural* gene which will define which proteins will enter the cell's *cytoplasm* from the *environment*; another type of *structural* gene, the *environmental gene* determines which proteins will be present in the cell environment (as maternal factors); finally, a *behavioural gene* is also a type of *structural* gene which contains operator and cell behavioural regions.

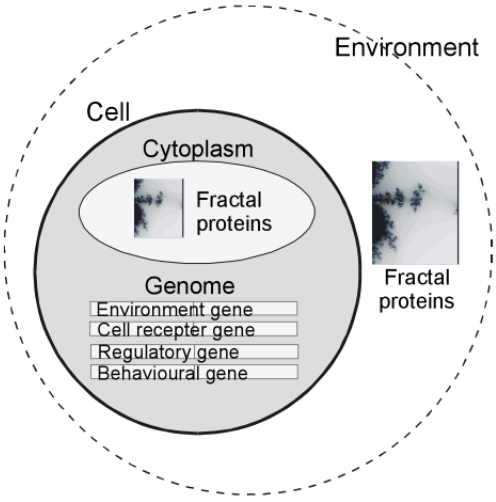


Figure 2.12: From [Krohn et al., 2009]: Representation using fractal proteins.

Every iteration, each of the cells will perform a developmental process comprising the following steps[12]: 1) the environmental genes are expressed and its merged fractal proteins shape is calculated; 2) Cell receptor genes are expressed and then used to mask the new environmental proteins into the cytoplasm; 3) the regulatory genes are expressed (if their promoter matches a cytoplasm protein) and the corresponding proteins are added to the cytoplasm; 4) the behavioural genes are then used in the same form as regulatory genes but specifying a cellular function instead; 5) finally, the concentrations of all proteins in the cytoplasm are updated.

Each gene has a promoter and a coding region. The promoter is used to match the gene with fractal proteins. The coding region comprises a region for the product fractal protein's parameters and another for the type of the gene. Each gene may be multifunctional, i.e., the *type* can be any combination of *environment*, *receptor*, *behavioural* or *regulatory*, resulting in the gene being expressed with different functions through the developmental process. This feature eliminates problems with positional dependence enabling also variable length genomes.

The problem of finding the digits of π was addressed in two different ways: through the binary representation (which also does not contain patterns) and through the values of the digits, both using an incremental fitness approach. According to the author the reported results are "impressive", as very few iterations are needed to successfully reach solutions up to 15 decimal places.

The model was later adapted to evolve behavioural genes able to control an inverted pendulum, in [Krohn and Gorse, 2010, Krohn and Gorse, 2012].

Extended Artificial Regulatory Network

The Artificial Regulatory Network (ARN) is a model proposed by [Banzhaf, 2003] which replicates some aspects of biological regulatory networks and has been shown to have potential for application in GP (this model is described in detail in Section 3.1).

To use the ARN model as a representation formalism for genetic programming one needs to define what are the inputs and what are the outputs. For that purpose Banzhaf's model was extended in two directions, in [Nicolau et al., 2010]. First, some extra proteins, not produced by genes but contributing to regulation, were introduced and act as

[12]For the sake of simplicity some details on fractal proteins manipulation are omitted. Please refer to the original publications for a complete description of the implementation.

inputs. Second, the genes were divided into two sets, one producing proteins that are used in regulation (i.e., transcription factors), and a second one with proteins without regulatory function which are used as outputs. These two types of genes are distinguished by having different promoters (Fig. 2.13).

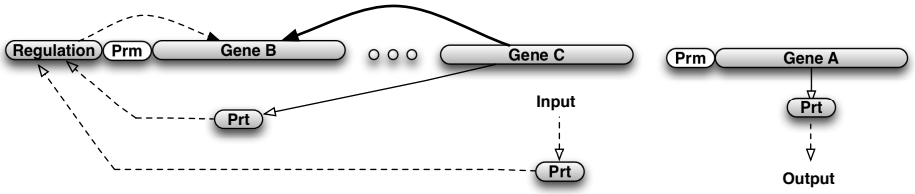


Figure 2.13: The modified ARN

This model was tested with the classic control problem known as the pole balancing problem. At every n iterations the output is read by choosing the output protein with highest concentration (or alternatively, the biggest gradient) and feeding this action to the controller. Consequently with a new iteration at the controller simulation, the extra proteins concentration will be modified, introducing feedback from the controller's state into the system, which will have n iterations to stabilise until a sampling is performed. The results reported are very competitive with the ones found in literature.

This representation has further been used in different contexts. In [Nicolau et al., 2012] the model is used as a predictor for financial time-series. In [Murphy et al., 2012] it is used again to control an inverted pendulum. In this case the controller is still the dynamic behaviour of the regulatory network by means of protein expression as described earlier, but with a different mapping applied to the output proteins, by means of tree-adjunct grammars. Finally, in a different framework by [Cussat-Blanc et al., 2011], the model was used with the extra-proteins serving as morphogens to differentiate cell behaviour in a grid, colouring the cell based on the location of the active genes towards a *French flag*.

The Artificial Epigenetic Network

[Turner et al., 2013] propose a model of artificial epigenetic networks (AEN), including a layer of epigenetic control, using the same representation of the GRN layer. They apply the model to evolve controllers for the coupled inverted pendulum problem showing significant performance improvement.

The model is composed of a set of genes, each coding its properties using real values (including the expression function properties, and the neighbourhood). Contrary to the ARN model presented in the last section, this proposal uses an indirect reference space to represent the connections between genes (based on a real value as the *identifier* - or position - of the gene, and another real value as the *proximity*, defining the neighbourhood of the gene). The authors have demonstrated that this scheme presents advantages in terms of the evolvability of the underlying structure. Moreover, it provides an efficient way of evaluating the dynamic networks in run-time. The architecture handles inputs and outputs through designated genes in the network, respectively the lowest and highest (ordered by the identifier). The input values are mapped into the expression values of the lowest genes, each gene updates its expression value (ordered by the identifier), and the output is extracted from the expression values of the highest genes.

The epigenetic layer is coded in the same way as the genetic one, but the elements are called epigenetic molecules. The real values in these molecules define an identifier and neighbourhood in the same way as with the genes, as well as the activation function properties. In this case, there is not an expression value, each iteration the result of the function is calculated and it is used as a binary switch, activating or deactivating the molecules. The active molecules inhibit the genes that fall in their inhibition neighbourhood (also coded by an identifier and proximity real values) by setting their expression value to 0.0 and effectively removing them from the network. The evolutionary process is run using a typical GA, since the AENs are arrays of real values.

The authors found that the evolved controllers partition the gene network using the dynamic (de)activation of the epigenetic molecules. A set of genes is used to balance the pole until the upright position, and a different one is activated to keep it there. This is a plausible behaviour according to what is known from biological systems (see Section 2.1), making this a promising model to use in more complex problems.

3

The Regulatory Network Computational Device

We have seen in the previous chapter that the depth of the knowledge about biological systems has largely increased since Darwin. In contrast, the paradigms used in Evolutionary Computation (EC) are typically centred on the evolutionary principles from the times of the modern synthesis. Nevertheless, some researchers have started exploring the application of this new biological knowledge to the representations of the evolutionary algorithms (see Section 2.3), for instance, through the implementation of indirect encodings, and the introduction of development analogies.

Despite the many proposals of more or less plausible computational models for diverse evolutionary and developmental mechanisms, particularly for genetic regulatory networks (GRN) (see Section 2.2), few attempts have been made to incorporate these models into Artificial Evolution (AE). As mentioned in Chapter 1, a paradigm shift from AE to Computational Evolution (CE) is recommended by some researchers, to be accomplished by incorporating algorithmic analogues of our current knowledge on natural evolution. In particular, (a) the influence of the environment, (b) gene expression, and (c) development. As demonstrated by some of the examples of developmental models presented before (Section 2.3.2), the proposed paradigm shift may have (slowly) started already.

The Regulatory Network Computational Device (ReNCoDe) [Lopes and Costa, 2011a, Lopes and Costa, 2011c, Lopes and Costa, 2011b, Lopes and Costa, 2012, Lopes and Costa, 2013a], is the author's contribution to integrate gene regulatory networks into a general problem solving framework in EC. This architecture uses the Artificial Regulatory

Network (ARN) proposed by [Banzhaf, 2003] as the genotype of the individuals. This model mimics not only topological properties of natural GRN, but also the expression behaviour of the proteins coded in the genome (Section 3.1). Moreover, the genome is linearised, a property common in the latest representations in the field. Given the ARN of the individual (composed of a network of genes that each map a protein), the network graph is reduced and a GP-like program graph is extracted from the network (Section 3.2). In order to apply this framework to problems with inherent recurrence a variant of the algorithm was created, where feedback mechanisms are allowed (Section 3.3).Finally, biologically inspired asexual and sexual variation operators were also implemented to deal with the genotypic representation of the ARN (Section 3.4).

3.1 The Artificial Regulatory Network Model

The Artificial Regulatory Network (ARN) [Banzhaf, 2003] is an attempt to incorporate regulatory mechanisms between the genotype and the phenotype. There are no other products, i.e., DNA, and processes in between these two levels. The genome has fixed length and is constructed by simple duplication with mutation events. Regulation between genes is a mediated process, achieved by means of a binding process between proteins (i.e., transcription factors) and special zones in the genome that appear upstream of the promoter of a gene. The remaining of this section will describe this with more detail.

Genome and Gene Expression

The ARN model uses a binary genome and implements a simple algorithm to transcribe and then translate it into proteins. The genome can be generated randomly or by a process of duplication with mutation, also called *DM*, that is considered the driving force for creating new genes in biological genomes and has an important role in the growth of gene regulatory networks [Teichmann and Babu, 2004]. In the latter case we start with a random 32-bit binary sequence, that is followed by several DM episodes. As we will see later the number of duplications is an important parameter. So, if we have 10 duplication events then the final length of the genome is $2^5 \times 2^{10} = 32768$. The mutation rate is typically of 1%. Each gene in the genome is divided in several regions, namely a regulatory site, the promoter, and the coding region of the gene itself. The first 32 bits

of the regulation zone are the enhancer site, while the following 32 bits are the inhibitory site. The promoter is located downstream and has the form $XYZ01010101$.The first 24 bits (the sequence represented by XYZ) can be either 0 or 1, while the last 8 bits are fixed. This way, the probability for a promoter to occur is $2^{-8} = 0,39\%$. The idea is to model what happens in nature, where we have a small consensus sequence where the RNA polymerase binds (in our case 01010101), inside a larger promoter. A gene is a set of five 32-bit long consecutive sequences, i.e., a 160-bit string. The choice of the method to obtain the genome, including the values for the parameters, is guided by what happens in the natural world.

Figure 3.1 gives an idea of the representation. P is the promoter region, that indicates the beginning of the gene; $G1$ to $G5$ are the five parts of a gene; E is the gene activation binding site (enhancer) for the protein, and H is the repression binding site for the protein (inhibitor).

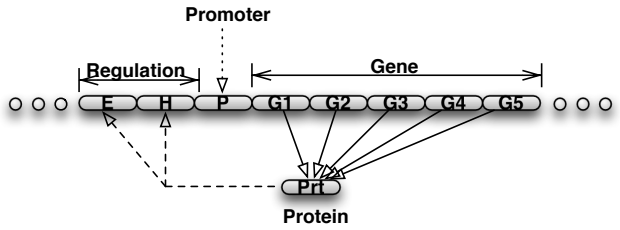


Figure 3.1: A genome element in the ARN.

The genotype - phenotype mapping is defined by expressing each 160-bit long gene, resulting in a 32-bit protein. A gene expresses a protein by a majority rule: if we consider a gene, for example G^m, divided into 5 parts of size 32 each, G^m_1 to G^m_5, at position i, say, the protein's bit will have a value corresponding to the most frequent value in each of these 5 parts, at the same position, i.e.,

$$P^m_i = majority(G^m_{ki}, \quad \forall k = 1, \ldots, 5), \quad \forall i = 1, \ldots, 32$$

Figure 3.2 shows a simple illustrative example.

Regulation

Genes interact mediated by proteins, which bind to the regulatory region of each gene. If, say, gene $\mathbf{A}$ expresses protein $\mathbf{p_A}$ and that protein contributes to the activation of

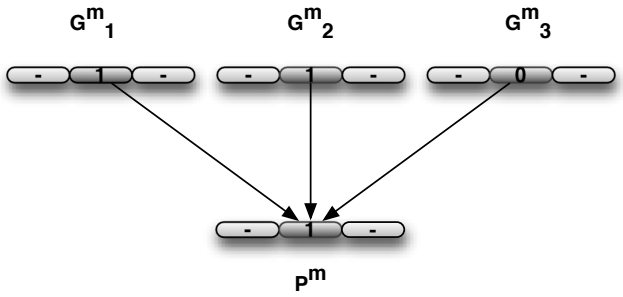


Figure 3.2: The majority rule: a simple example for a gene with three regions of size three and the corresponding protein. Here just the process for the second position.

gene **B**, we say that gene **A** regulates **B** (see Fig. 3.3).

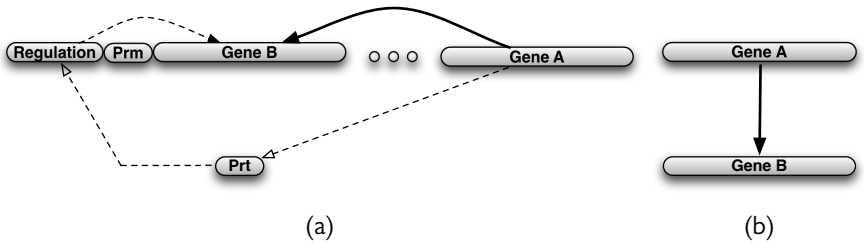


Figure 3.3: Gene - Protein - Gene interaction

Notice that in order for a link to exist between any two genes, the concentration of the corresponding protein must attain a certain level, and that depends on the strength of the binding. The strength of the binding is computed by calculating the degree of complementarity between the protein and each of the regulatory regions, according to formula 3.1:

$$x_i = \frac{1}{N} \sum_{\substack{0<j\leq N \\ j\neq i}} c_j \exp^{\beta(\mu_{ji}-\mu_{max})} \tag{3.1}$$

where x_i represents the binding strength of the enhancer (e_i) or the inhibitory (h_i) region, N is the number of proteins, c_j the concentration of protein j, μ_{ji} is the number of bits that are different in the protein j and in each of the regulation sites of protein i (r_i), that is,

$$\mu_{ji} = \sum_{k=1}^{32} j_k \oplus r_{i_k} \tag{3.2}$$

μ_{max} is the maximum match achievable, and β is a scaling factor. The production of a protein over time depends on its concentration, which in turn is a function of the way each protein binds to that gene's regulatory regions. It is defined by the differential equation

$$\frac{dc_i}{dt} = \delta(e_i - h_i)c_i \tag{3.3}$$

where e_i and h_i are defined by equation 3.1, and δ is a scaling factor.

Computational Device

Using this process we can build for each genome the corresponding artificial gene regulatory network. These networks can be studied in different ways. We can be concerned with topological aspects (i.e., to study the degrees distribution, the clustering coefficient, small world or scale free, etc.) or the dynamics of the ARNs (i.e., attractors, influence of the protein-gene binding threshold) [Nicolau and Schoenauer, 2009, Kuo et al., 2006]. This is interesting, but from a problem-solving perspective what we want is to see how the model can be used as a computational device. In order to transform an ARN into a computational problem-solver we need to clarify what we put in the system (including the establishment of what is the relationship with the environment) and what we extract from the system. At the same time we need to define the semantics, that is, the meaning of the computation in which the network is engaged. Finally, and as a consequence of the points just identified, it is also fundamental to determine if we are interested in the input/output relationship or if what we want is just the output. A solution for the latter situation was proposed in [Kuo et al., 2004] in the context of optimisation problems. The idea is to define (randomly) two new contiguous 32-bit sequences in the genome. The first one being a new inhibitory site (h_i), and the second one a new activation site (e_i). All generated proteins can bind to these sites. The levels of activation and inhibition can be computed as before (see equation 3.1), but there is no gene (thus no protein) attached (see figure 3.4).

The state of this site is just the sum of all bindings (see equation 3.4) and is defined as

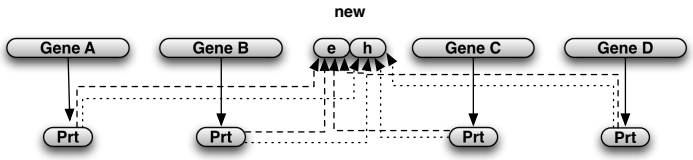


Figure 3.4: Fabricating an output

the output. This additional binding is thus a method to extract a meaning from the variation of the proteins' concentrations over time, and was used to evolve simple functions, like the $sin(t)$.

$$s(t) = \sum_i (e_i - h_i) \tag{3.4}$$

As mentioned before, in order to use the model as a representation formalism for GP one needs to define what are the inputs and what are the outputs. For that purpose the ARN model was extended in two directions [Nicolau et al., 2010]. First, some extra proteins, not produced by genes but contributing to regulation, were introduced and act as inputs. These have the following distinct signatures:

00000000000000000000000000000000
00000000000000001111111111111111
11111111111111110000000000000000
11111111111111111111111111111111

Second, the genes were divided into two sets, one producing proteins that are used in regulation (i.e., transcriptional factors), and a second one with proteins without regulatory function which are used as outputs. These two types of genes are distinguished by having different promoters: $XYZ00000000$ and $XYZ11111111$, respectively (see figure 3.5). Moreover, the differential equations that model the protein concentrations were also adapted to accommodate the new proteins type and behaviour.

This representation has further been used in different contexts. In [Nicolau et al., 2012] the model is used as a predictor for financial time-series. In [Murphy et al., 2012] it is used again to control an inverted pendulum. In the latter case the pendulum is still controlled by the dynamic behaviour of the regulatory network (by means of protein expression as described earlier), but with a different mapping applied to the output proteins, this time using tree-adjunct grammars. Finally, in a different framework by [Cussat-Blanc

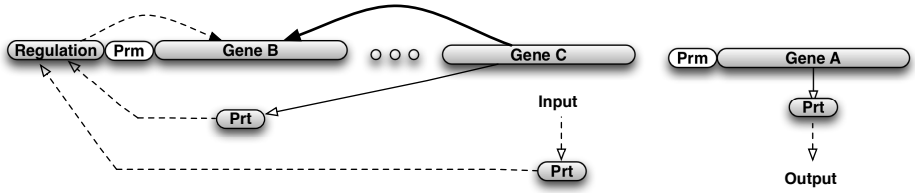


Figure 3.5: The modified ARN

et al., 2011], the model was used with the extra-proteins serving as morphogens to differentiate cell behaviour in a grid, colouring the cell based on the location of the active genes towards a *French flag*.

In the present work a different path is followed, that does not take into account the proteins' concentration but only the regulatory network graph. Both the original and the extended ARN model formulations will be used in different contexts, the former when the problems only require one output (either with or without recursion) and the latter when multiple outputs are required. The model presented in [Lopes and Costa, 2012], in which the ARN architecture is used as the genotypical representation for a new computational model will be described in detail in the following sections.

3.2 Extracting Programs from Regulatory Networks

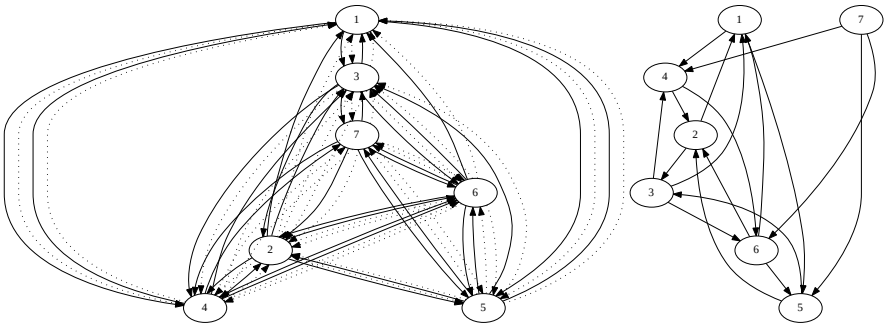
The main idea of the Regulatory Network Computational Device - ReNCoDe - is to simplify the ARN to produce a graph that can be computed similarly to a GP system. The networks resulting from ARN genomes are very complex, composed of multiple links (inhibition and excitation) between different nodes (genes). In order to extract a circuit from these networks they must first be reduced, input and output nodes must be identified, and one must ascribe a semantic to its nodes. In the simplest cases, the final product will be an executable feed-forward circuit.

Algorithm 1 shows the pseudocode for the reduction algorithm. First, every pair of connections - excitation (e) and inhibition (h) - is transformed into one single connection with strength equal to the difference of the originals (e-h) (line 2). Second, every connection with negative or null strength is discarded (lines 3-5). Finally, a directed graph is

built adding only the nodes with positive edges and the strongest edge between each pair of nodes (lines 7-13). This process is illustrated in Figure 3.6a (initial network) and 3.6b (after applying the algorithm).

Algorithm 1 Artificial regulatory network reduction.

```
1:  for all gene in network do
2:      replace bindings by edge (e-h)
3:      if (e − h) ≤ 0 then
4:          remove edge
5:      end if
6:  end for
7:  for all edge(i, j) in network do
8:      if edge(i, j) ≤ edge(j, i) then
9:          remove egde(i, j)
10:     else
11:         remove edge(j, i)
12:     end if
13: end for
```



(a) (b)

Figure 3.6: a) ARN example. The dotted and full edges represent, respectively, inhibition and excitation relationships between genes. The numbers are just identifiers. b) The same network after application of the Algorithm 1.

Next, the node with the highest input connectivity is chosen as the output and inserted into a *queue*. After this, the circuit is built backwards from the output function until the

terminals (nodes without input edges) are reached. The pseudo-code for this process is given in Algorithm 2.

Algorithm 2 Extracting a program from an ARN

```
1:  sort nodes by input-edge number
2:  add top-node to queue
3:  while queue is not empty do
4:     map top-node to function
5:     add function to program
6:     add input-edges to list
7:     remove top-node from queue
8:     for all input in list do
9:        if  no dependencies and not in program and not in queue then
10:          add input to queue
11:          remove input from list
12:       end if
13:    end for
14:    if queue is empty and list is not empty then
15:       sort list by input number
16:       add top-node to queue
17:    end if
18:    sort queue by input-edge number
19: end while
```

Each time a node is added to the program and mapped to a function, the corresponding inputs are added to a temporary list (line 7) and those without dependencies (that is, those that are not input to another node already in the *queue*) are added to the *queue* (lines 8-13) and ordered by input-edge number. By choosing the nodes with the most input-edges one reduces the dependencies of nodes in the *list* with the *queue*, avoiding more cycles. Nevertheless, it is possible to encounter cycles at some point of the process. When this happens the *queue* will be empty, and there is a deadlock in the *list* (every function is input to some other). To resolve this again the gene with highest connectivity is chosen (lines 14-17).

Each time a node is mapped (line 4), the inputs from nodes already in the program

are simply discarded resulting in state-less feed-forward program graphs (see Figure 3.7). This corresponds to line 1 in the pseudo-code for the mapping function, which is given in Algorithm 3.

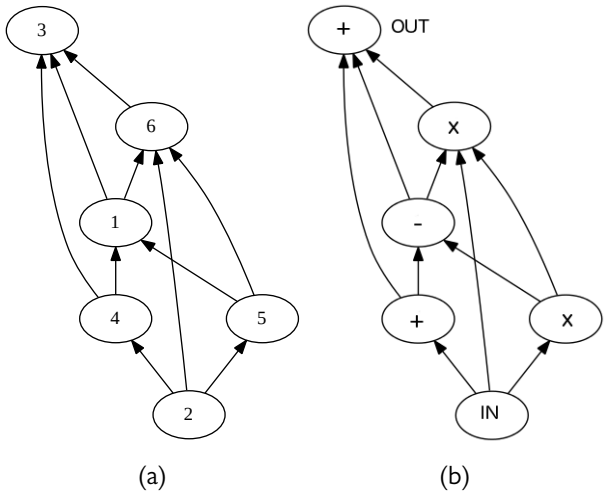


Figure 3.7: a) The executable graph extracted from the reduced network depicted in Fig. 3.6b. This circuit is obtained by choosing node 3 as the output and then successively adding the inputs of each node until the terminal nodes are reached. Note that one of the cycles in Fig. 3.6b is composed of the nodes 3, 4, and 2 and the connections from 3 to 4 and 2 are discarded. Functions take their inputs from the nodes where their in-edges come from. b) The same graph with the nodes mapped to functions/terminals.

To map nodes (i.e., genes) to functions and terminals the gene-protein correspondence is used. The protein's signature is subjected to a majority vote process, obtaining the function/terminal index (each protein codifies a function/terminal).

As an example, to code the function set $\{+, -, \times, \div\}$ only two bits are necessary. The protein's signature is split into sixteen two-bit chunks. Then we obtain the function set index (two bits) by applying the majority vote rule over each bit (two positions) of the sixteen chunks (since the number of chunks is even, in case of a tie the result holds 1). If the function set size is not a power of two, then dummy functions[1] may be added to the set. If some determined problem has more than one input, for instance four, then the majority rule is applied over the terminal protein signature (its binary stream) in order

[1] A dummy function is an empty function that maintains the state of the program and does not influence the result.

Algorithm 3 Mapping nodes to program functions.

```
1: filter node-inputs already in the program
2: if node-inputs not empty then
3:     funindex ← apply majorityrule to protein-signature
4:     map node to corresponding function
5: else {node-inputs is empty}
6:     termindex ← apply majorityrule to protein-signature
7:     map node to corresponding terminal
8: end if
```

to define which input it corresponds to. Figure 3.8 exemplifies the majority rule applied over a protein signature for a function set with four elements.

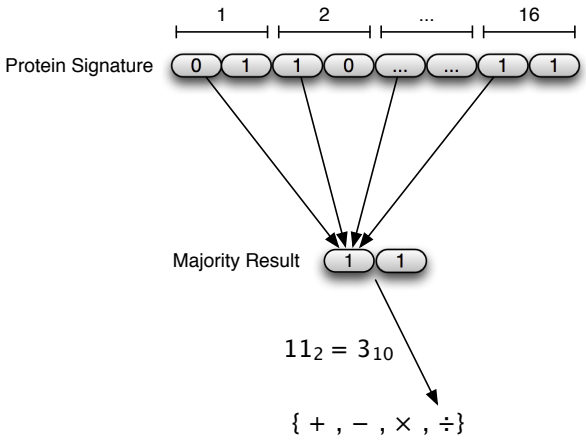


Figure 3.8: Example of the mapping of a protein's signature to a program function, by applying the majority rule over the bitstream. In the example the result is 11, which converted to the decimal base gives the index 3 on the function set.

Finally, the resulting program can be executed in a feed-forward fashion by propagating the output value of each node through the graph until the output is reached (similarly to a lazy-evaluation), or be recursively translated into nested function calls of a given language.

3.3 Using Feedback Connections

Feed-forward circuits may not be adequate for every type of problem. For instance, problems where memory is necessary or with inherent recursion may be approached differently. For that purpose the possibility of using feedback connections in the program's graph was added to the original model [Lopes and Costa, 2011c].

As mentioned in the previous section, in the original formulation of ReNCoDe each time a node is mapped to a function in the program (Alg. 2, line 4), the input edges from nodes already in the program are simply discarded (Alg. 3) resulting in state-less feed-forward circuits (Fig. 3.7). However, if one assumes that a node in the program holds its value until it is re-written, then these recurrent connections become useful and it is possible to map ARNs to state-full programs.

The difference is that when a node is mapped into a function in the program, if there are input-edges from nodes previously mapped these are not discarded. Instead they are marked as negative input-edges, as described in Algorithm 4. This results in a cyclic graph which when iterated is able to store information from the previous iterations (see Figure 3.9 for an example where this type of connection is represented by dotted edges).

Algorithm 4 Mapping nodes to program functions with feedback connections.

```
1:  for all input in node-inputs do
2:      if input in program then
3:          negate input
4:      end if
5:  end for
6:  if node-inputs not empty then
7:      funindex ← apply majorityrule to protein-signature
8:      map node to corresponding function
9:  else {node-inputs is empty}
10:     termindex ← apply majorityrule to protein-signature
11:     map node to corresponding terminal
12: end if
```

The marking serves the purposes of tracking and displaying the programs only. During program execution the negative sign is actually ignored. Reading a node yields its output

value, which is written only when the node is evaluated. In the case of (input) nodes upper in the graph, when the program is iterated these store the value from the previous iteration. In the case of the first iteration the functions return the default value (1.0).

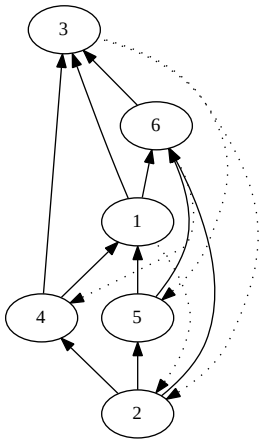


Figure 3.9: The resulting circuit from the reduced network depicted in Fig. 3.6b. The connections from node 3 to the other nodes are used as feedback, since node 3 was already in the circuit. Functions take their inputs from the nodes where their in-edges come from. The dotted edges represent feedback connections (negative input-edges).

3.4 Genetic Operators

The basic operator for ReNCoDe to be used with Evolution Strategies (ES) is the *bit-flip* mutation, which based on an uniform distribution will randomly flip individual bits of the genome. This allows the corresponding networks to be modified but the genome's length is fixed.

Variable length genome's are known to improve evolvability with different representations [Trefzer et al., 2011]. The ARN representation has implicit variable length, since the number of coded genes may vary with mutations. With the goal of making the length of the ARN genomes explicitly variable, three biologically inspired operators were developed and presented in [Lopes and Costa, 2012] to be applied to ARN genomes before *bit-flip* mutation, without taking into account the network elements (genes, promoters, regulatory sites). Besides these *bitwise operators*, *genewise operators* developed for the ARN will also be presented in this section. Finally, typical crossover operators (both

bitwise and *genewise*) will also be described.

3.4.1 Bitwise Operators

In order to increase the size of the noncoding regions of the genome, a *junk* operator was created. This introduces a stream of zeros at a random location in the genome (see Fig.3.10). Since promoters are not duplicated with this operator nor directly inserted, the number of genes will typically be lower, and may increase the number and size of neutral regions in the genome.

Inspired by the concept of asexual transposition in nature, a *transposon-like* operator was developed, that copies and pastes a random portion of the genome at a random location (Fig. 3.11), increasing the genome size.

In order to allow the genomes to be manipulated to smaller sizes as well as avoid indefinite growth, a *delete* operator was also implemented (Fig. 3.12). This removes a random genome part of the genome, being designed to use in conjunction with any of the operators presented before.

Finally, any of these operators can be applied with a predefined (fixed) portion length. The sensitivity of the model to this parameter will also be investigated (see Chapter 5).

To perform crossover of individuals three typical operators were implemented: one-point, two-point, and uniform crossover. These are the typical crossover operators used in EC [Eiben and Smith, 2003]. In a nutshell, these operators select one or more cutting points in the progenitors and create offsprings by swapping the genetic material between them.

3.4.2 Genewise Operators

Inspired by the operators used in [Crombach and Hogeweg, 2008], two *genewise* operators were developed. One is functionally similar to *transposon*, copies a random gene (regulatory sites, promoter, and coding region) to the end of the genome. When a complete gene is copied it can be inserted into any non-coding region of the genome, since the regulatory network is independent of gene position. The other removes a random gene from the genome. The copy example is presented in Figure 3.13.

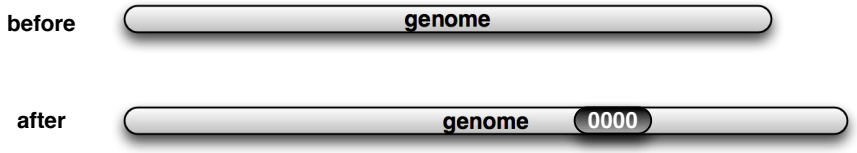


Figure 3.10: The effect of using an operator inspired by the concept of junk DNA, by inserting a stream of 0's into a random location on the genome.

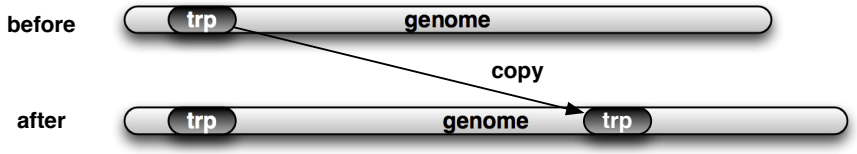


Figure 3.11: The effect of using a transposon-like operator on modifying the genome: the copy example.

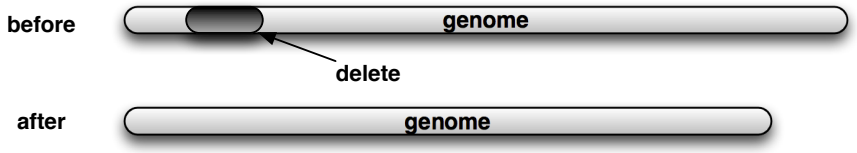


Figure 3.12: The effect of using a delete operator on the genome: a random portion is removed.

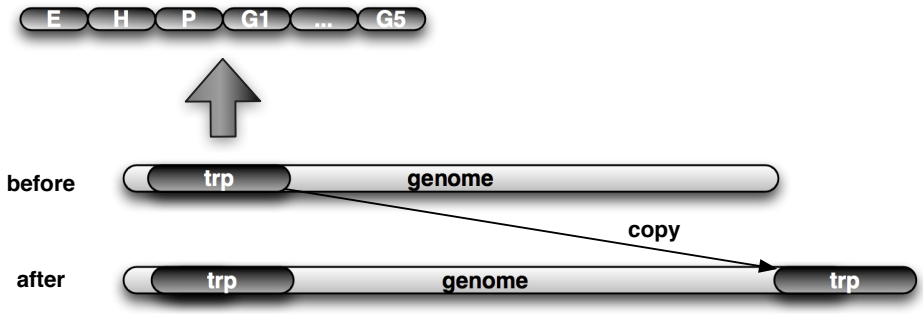


Figure 3.13: The effect of using a genewise operator on modifying the genome: the copy example.

A similar approach was used for the crossover operators. One-point, two-point, and uniform crossover were implemented, restricting the cutting points to the non-coding regions contiguous to the genes. This way the individuals exchange between them sets of (one or more) complete genes.

4

Benchmark Problems

The previous chapter presented a new computational model using artificial regulatory networks as the genotypic representation of the individuals, and clarified how to extract GP-like program graphs from those networks. Every new algorithm needs testing, and for that purpose benchmark problems are used.

Benchmarks come in different flavours and from several problem domains. The ultimate solver would be able to solve problems from every class possible. However, all algorithms are limited to some classes of problems or perform badly outside the problem domain they were targeted to (the *no free lunch* theorem). Nowadays there is a GP community-driven effort to find agreement on appropriate benchmarks for the field. The latest progress includes a complete survey of the problems used in the field [McDermott et al., 2012] and an inquiry to the community researchers [White et al., 2012]. This effort resulted so far in a list of problems that should not be used, and proposes some alternatives in the corresponding problem domains. These guidelines were taken into account when choosing the tests for the model described in the previous chapter.

The basic ReNCoDe is adequate for the traditional GP benchmarks, and was tested using four problems: the harmonic curve regression, a polynomial regression, the inverted pendulum, and the Santa Fe ant trail (Section 4.1). Allowing feedback connections during the program extraction, allows one to solve problems that have a recursive or iterative nature like the Fibonacci sequence, the squares generator, the even n-bit parity, or a factorial-like sequence (Section 4.2).

4.1 Without Recursion

The traditional benchmark problems in GP require a program that is able to deal with one or more inputs, and that outputs a single value to be interpreted. There are several domains of application and countless benchmarks. In this section a representative set is presented, which was used to test ReNCoDe.

4.1.1 Symbolic Regression

Symbolic regression is perhaps the most acclaimed and studied problem in GP. It is easily implemented and tested, allowing the quick investigation of many aspects of the algorithms.

Many different functions have been used by researchers as the target. For the purpose of testing ReNCoDe two instances were chosen: a traditional polynomial regression, and the harmonic number. These will be described in the remaining of this section.

Polynomial Regression

Traditionally in GP the goal of symbolic regression is to fit a polynomial expression to a set of points over some pre-determined range.

The function set is composed by the arithmetic operators $\{+, -, \times, \div\}$, while the terminal set has only the input $\{x\}$. Protected division is used, returning 1.0 whenever the denominator is 0. Usually the arithmetic operators have a fixed arity of 2. In ReNCoDe, however, the arity is defined by the regulatory network structure, in order to allow higher function reusability.

The typical expression that is used as the target was defined by [Koza, 1992] and corresponds to the following polynomial of sixth degree:

$$f(x) = x^6 - 2x^4 + x^2$$

The function is sampled over the range $[-1, 1]$, with a step $s = 0.1$. The fitness function used is the sum of the absolute errors over the sampled points, and the evolutionary run stops when either an individual is found for which this value is inferior to 10^{-3} or the

maximum number of evaluations is reached. For the sake of reference, [Miller and Thomson, 2000] report a success rate of 61% in experiments limited to 80000 evaluations.

The Harmonic Number

One of the new problems identified by the authors of the supra-cited survey in the symbolic regression domain, was the harmonic number (defined in Equation 4.1). This series can be approximated using the asymptotic expansion presented in Equation 4.2, where γ is the Euler's constant ($\gamma \approx 0.57722$).

$$H_n = \sum_{i=1}^{n} \frac{1}{i} \tag{4.1}$$

$$H_n = \gamma + ln(n) + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4} - \dots \tag{4.2}$$

This problem is particularly interesting because there is not only an interpolation task (during the evolutionary process), but also a generalisation task with the evolved solution. The goal is to evolve a program that approximates the harmonic number, given an input belonging to the set of natural integers from 1 to 50. Upon the evolutionary phase the best solution is tested for generalisation over the set of natural integers from 1 to 120. This problem was studied in the context of GP by [Streeter, 2001], and the proposed approach was able to rediscover the asymptotic approximations, with absolute error sum in the order of 10^{-2} .

The function set used is composed of $\{+, \times, \mathit{reciprocalsum}, \ln, \mathit{sqrt}, \cos\}$. The (protected) division function is not explicitly available. Instead, it is replaced by the *reciprocalsum* function, which returns the sum of the reciprocals of each input. If one of the inputs is zero the result is 10^6. All non-terminal nodes have variable arity in ReNCoDe, thus the unary functions have to be adapted. A node mapped to any of $\{\ln, \mathit{sqrt}, \cos\}$ with more than one input, will first sum the corresponding inputs and then apply the function. The terminal set is composed only of $\{n\}$. Every function is protected: in case of overflow or impossible values provided as inputs the return value is 10^6.

The fitness function used in this work was the Mean Squared Error (MSE) between the output of the individual and the target function. A linear-scaling approach to fitness is used [Keijzer, 2003]. Given $y = gp(x)$ as the output of an expression evolved by GP on the

input data x, a linear regression on the target values t can be performed using Equations 4.3 and 4.4, where $\bar{y}$ and $\bar{t}$ denote respectively the average output and the average target value. These equations calculate the *slope* and *intercept* of the set of outputs y, minimising the sum of squared errors between t and $a + by$ (with a different from 0 and b different from 1). With the constants a and b calculated by this simple regression, all that is left to the evolutionary run is to find a function with the correct shape, using as fitness the modified MSE presented in Equation 4.5.

$$b = \frac{\sum[(t - \bar{t})(y - \bar{y})]}{\sum[(y - \bar{y})^2]} \tag{4.3}$$

$$a = \bar{t} - b\bar{y} \tag{4.4}$$

$$MSE(t, a + by) = \frac{1}{N}\sum_{i}^{N}(a + by - t)^2 \tag{4.5}$$

It is worth mentioning that individuals who generate a constant output for every input are considered invalid, since the denominator in Equation 4.3 would result in 0, and thus the fitness returned is 10^6. By avoiding these individuals this linear scaling technique offers great advantage, since in our previous experiments with various symbolic regression problems, it was noticed that individuals generating a constant output correspond many times to local minima and many generations may be necessary to step out.

During the generalisation testing the constants a and b[1] are used to compute the output of the individual over the set of natural integers from 1 to 120, and compare with the target result. In this case, the fitness measure used was the Normalised Root Mean Square Error (*NRMSE*) as defined in [Keijzer, 2003] (Equation 4.6). The result reported for this function was $NRMSE = 1\%$.

$$NRMSE = 100\% \times \frac{\sqrt{\frac{N}{N-1}MSE}}{\sigma_t} \tag{4.6}$$

[1] Determined during the evolutionary phase

4.1.2 Artificial Ant

Imagine a 2D toroidal grid of size 32x32, laying on it a specific trail of 89 food pellets with a few gaps in between (Figure 4.1). Then imagine an artificial ant placed at the upper left corner of the grid facing east, whose goal is to collect the food pellets along the trail. The problem consists in evolving a controller for the artificial ant that successfully collects the food pellets in a limited number of actions.

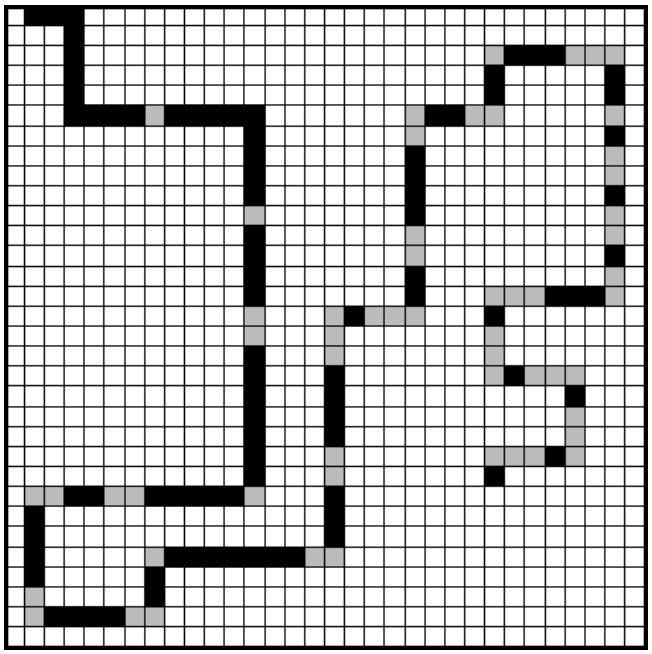


Figure 4.1: The Santa Fe trail for the Artificial Ant problem. The black squares represent the food pellets, and the grey squares the gaps in the trail.

The function set is composed of {*if-food-ahead*, *progn*} and the terminal set consists of {*left*, *right*, *move*}. The function *if-food-ahead* allows the ant to check whether there is food in the cell directly in front of it and perform some action (or group of actions) according to this result. With *progn* the ant will perform a set of consecutive actions, depending on the arity of the function. This is the replacement for the *prog2* and *prog3* used originally, with variable arity defined automatically by the network. The terminals allow the agent to turn 90 degrees left or right without moving, and to move to the adjacent cell it is facing. If a cell contains a food pellet it is immediately eaten when the ant stands on it.

Using the described functions and terminals a foraging strategy is built and each ant

is given a determined number of steps, 600 in this case, to apply it repeatedly searching for the 89 food pellets available in the trail. The fitness function used for this problem is the remaining number of food pellets, after the foraging strategy is applied.

The best result found in the literature is reported by [Christensen and Oppacher, 2007], and fairly compared to several other approaches in the literature. The reported computational effort for their best attempt is $CE = 20696$ [2]. Note that the results discussed in Section 6.1 do not use the CE and thus are not directly comparable, but considering that the success rate for ReNCoDe is 100% such comparison should not be harmful.

4.1.3 The Inverted Pendulum

The inverted pendulum is a typical control benchmark, in which the goal is to keep a pole (the inverted pendulum) balanced on top of a cart, and the cart within the track boundaries. The controller actuates on the cart by applying a constant force F in either direction [Whitley et al., 1993].

Typically, to evolve or train a controller for the cart the following four input variables are used:

$x \in [-2.4, 2.4]m$, the position of the cart relative to the center of the track;
$\theta \in [-12, 12]°$, the angle of the pole with the vertical axis;
$\dot{x} \in [-1, 1]ms^{-1}$, the velocity of the cart;
$\dot{\theta} \in [-1.5, 1.5]°s^{-1}$, the angular velocity of the pole.

During evaluation of the controllers the system is simulated using the equations of motion presented below (Eq. 4.7 and 4.8), where the gravity is $g = 9.8ms^{-2}$, $l = 0.5m$ is the pole's length, $m = 0.1Kg$ is the pole's mass, $m_c = 1.0Kg$ is the mass of the cart, and $F(t) = \pm 10N$ is the force applied by the allowed commands (push left or right).

[2] Computational Effort (CE) is an estimate of the number of individuals that need to be evaluated so that the cumulative probability of success reaches some level (usually 99%). Note that very large populations of runs should be considered. The interested reader may consult [Koza, 1992, Koza, 1994] for details and weaknesses of this measure.

$$\ddot{\theta}(t) = \frac{g \sin\theta(t) - \cos\theta(t) \left(\frac{F(t) + ml\dot{\theta}(t)^2 \sin\theta(t)}{m_c + m} \right)}{l \left(\frac{4}{3} - \frac{m \cos^2\theta(t)}{m_c + m} \right)} \tag{4.7}$$

$$\ddot{x}(t) = \frac{\frac{F(t) + ml\dot{\theta}(t)^2 \sin\theta(t)}{m_c + m} - ml\ddot{\theta}(t) \cos\theta(t)}{m_c + m} \tag{4.8}$$

The timestep used in the simulation is $\Delta t = 0.02s$, ending when either the cart reaches the track boundaries ($x = \pm 2.4m$), or the pole falls ($|\theta| > 12°$).

The function set used to build solutions to this problem with ReNCoDe is $\{+, -, \times, \div\}$. The terminal set is composed of the input variables described earlier $\{x, \theta, \dot{x}, \dot{\theta}\}$. Using these building-blocks a controller is constructed whose output is translated into a push to the left if negative, or to the right if positive.

Each individual was tested with a randomised initial position, using the same fitness function as in [Nicolau et al., 2010]:

$$F(x) = \frac{120000}{\text{number\_of\_successful\_steps}}$$

The evolutionary run terminates when an individual is found that successfully balances the pole for 120000 steps for each trial, or the maximum number of evaluations is reached. After the evolutionary process the controller is tested for generalisation over $5^4 = 625$ trials (this corresponds to combining the four input variables, with five different rates for each {0.05, 0.275, 0.5, 0.725, 0.95}), during 1000 steps, as described in [Whitley et al., 1993, Nicolau et al., 2010]. The best results known for this exact problem specification to the best of the author's knowledge are those reported in [Nicolau et al., 2010] (transcribed in Table 4.1).

Table 4.1: Summary of the best generalisation results for the cart-pole problem presented in [Nicolau et al., 2010], by number of successful start positions.

#Runs	50
Best	422
Mean	202.18
Std. Dev.	110.01

4.2 With Recursion

There are many problems that require some kind of memory embedded in the representation, in order to be possible to find a correct solution. The task in these problems is to evolve a solution using a small subset of test-cases, and verify its generalisation abilities using a different, and broader subset. As these problems have infinite test cases, it is not possible to computationally evaluate the solution correctness for the complete domains. Nevertheless, given that a solution passes the generalisation test, it can be manually inspected and formalised in order to demonstrate its generality.

4.2.1 Even Parity

The goal of this problem is to evolve a boolean function (or circuit) that takes a binary string as input and returns a single output which indicates whether the number of 1s in the string is even (0) or odd (1). The typical function set is $\{and, or, nand, nor\}$, using the input bits $\{x_1, x_2, \ldots, x_n\}$ as the terminal set.

It has been recognised as a difficult problem for evolutionary systems by different authors [Koza, 1992, Ferreira, 2002]. The traditional GP method is to directly evolve circuits (represented by trees). This approach does not scale well though, for instance 5-bit parity solutions are usually very difficult to evolve, and many times unsuccessful [Koza, 1992]. There have been improvements to this representation, amongst others the use of automatically defined functions [Koza, 1994], but still were not tackling n-bit parity.

Modern developmental systems have been proposed that solve the n-bit parity problem. In Self-Modifying Cartesian Genetic Programming (SM-CGP), the solutions are programs that construct circuits that modify themselves during execution [Harding et al., 2009b]. In [Kuyucu et al., 2009a] an artificial development system is evolved that also is capable of growth to generate a circuit that outputs the parity bit of any binary string. In the latter case however, the function set used was composed of multiplexers only, making the task easier. Other examples worth mentioning are the use of adaptive grammars [Wong and Mun, 2005], and the use of specific recursive functions [Wong and Leung, 1996].

When the aim is to find a general solution one possible approach is to use direct connections to the output of the previous evaluation or to iterate over the bits of the

binary input string, producing solutions suitable for any input size [Hohil et al., 1999, Harding et al., 2010]. Particularly, in [Harding et al., 2010] (the second version of SM-CGP), using CGP with functions that can directly modify the phenotype, the authors incrementally evolve solutions which through a developmental process generate parity checkers for each input stream length. To solve the input scalability issue special input pointers were designed which in successive evaluations allow to access different bits of the input. In this work they experimented with both the typical function set, and a more complete set, with several composed boolean functions. As described in Sect. 3.3 it is possible to use feedback connections in ReNCoDe, creating state-full circuits. Making use of this feature circuits which iterate over the input string bits will be evolved, similarly to recursive approaches, generating the parity bit as output.

The 3-bit even parity problem is used as the fitness function. That is, the evolutionary process halts when a solution that generates the correct parity bit for the eight (2^3) input combinations is found. If unsuccessful, it terminates when the maximum number of evaluations is reached. After the evolutionary process finds a solution for the 3-bit parity it is tested for generalisation, to a maximum of 24-bit input streams. Moreover, the evolved graph can be translated into boolean algebra allowing a formal analysis of the program's generalisation capabilities.

Although the results are not directly comparable amongst the approaches mentioned before, the best performance figures reported are presented for reference in Table 4.2.

Table 4.2: Summary of the n-bit parity results reported in the literature

Approach	[Harding et al., 2010]	[Wong and Mun, 2005]
#Runs	50	N.A.
Successful Runs (%)	100	90
General Solutions (%)	100	100
Max. #Evaluations	50K	50K
Avg. #Evaluations	150721	N.A.

4.2.2 The Fibonacci Sequence

The Fibonacci sequence is a recursive sequence that is defined by the following set of equations:

$$F(0) = 0 \tag{4.9}$$

$$F(1) = 1 \tag{4.10}$$

$$F(n) = F(n-1) + F(n-2) \tag{4.11}$$

This sequence has many real-world applications, for instance, in financial markets's analysis and computer algorithms. Moreover, it can be found in nature in diverse forms, such as the branching in trees or the arrangement of a pine cone.

This problem was first solved in genetic programming using recursive tree structures [Koza, 1992]. The results found in the literature show that obtaining general solutions is not an easy task. Most of the approaches take a very high number of evaluations to find a solution and are not completely effective. Moreover, the generalisation is usually poor, although some approaches managed to obtain good generalisation results[3].

The approach followed with ReNCoDe is to evolve circuits using the arithmetic operators as the function set $\{+, -, \times, \div\}$ and $\{0, 1\}$ as the terminal set. Protected division is used, returning 1.0 whenever the denominator is 0.

The program graphs are iterated to produce the sequence elements. The fitness function is the amount of correct numbers for the first ten elements of the Fibonacci sequence, in contrast to other approaches where the first 12 or the first 50 elements are used for training [Harding et al., 2009a]. The evolutionary process ends when the first ten elements are correctly generated or when the maximum amount of evaluations is reached.

Finally, the circuits are tested for generalisation over the first 74 elements of the Fibonacci sequence, in order to compare with the results reported in [Harding et al., 2009a]. The results for SM-CGP are transcribed in Table 4.3, although the approaches are not directly comparable. SM-CGP is a developmental approach, 12 elements of the sequence were used as fitness measure in that work, and the function set has more functions than the typical arithmetic operators.

[3] For a good summary of these results please refer to [Harding et al., 2009a].

Table 4.3: Summary of the Fibonacci sequence results for SM-CGP [Harding et al., 2009a], based on 287 runs. The first 12 elements of the sequence are used as fitness, and `{0,1}` as the starting condition.

Successful Runs (%)	89.1
General Solutions (%)	88.6
Avg. #Evaluations	1019981

4.2.3 The Squares Sequence

The sequence of squares is a sequential regression problem, where the target function x^2, over the non-negative integers, is to be evolved. The terminal set is composed of $\{1.0\}$. The particularity of this problem is that the function set is composed of only $\{+, -\}$. This limited function set only allows the regression of linear functions, and so this problem would be impossible to solve with traditional GP methods[4]. Based on the use of self-modification functions in the phenotype, different developmental approaches have been proposed that solve this problem with success, amongst them [Harding et al., 2009a].

In this problem we take advantage of the feedback connections in ReNCoDe to obtain circuits that, when iterated, will produce the correct squared value for the current iteration. Similarly to [Harding et al., 2009a] (which results are transcribed in Table 4.4) the evolutionary process tries to find a circuit that generates correctly the first ten terms of the sequence. Once a correct solution is reached the circuit is tested over the first hundred terms of the sequence to assess the generalisation capabilities of this approach.

Table 4.4: Summary of the squares' sequence results for SM-CGP [Harding et al., 2009a], averaged over 110 runs.

Successful Runs (%)	N.A.
General Solutions (%)	84.3
Min. #Evaluations	392
Avg. #Evaluations (Std. Dev.)	141846 (513008)

[4]Even if x is provided as input to the program, the number of operations necessary to generate the squared value varies with x itself, since the product is not available.

4.2.4 Modified Factorial

The modified factorial is a geometric sequence defined as $F(n) = k * F(n - s)$ where k and s are constants, respectively, a growth factor and a step. If $n < s$, then $F(n) = 1$ is used. For instance, for $k = 2$ and $s = 1$, the following sequence is generated:

$$F(0) = 1$$
$$F(n) = 2F(n-1) \ , \ n > 0$$

The recurrence relation can be solved by telescoping $F(n)$ for the first step instances, and one reaches the conclusion that:

$$F(n) = k^{\lfloor \frac{n}{s} \rfloor}$$

The function set is composed by the arithmetic operators $\{+, -, \times, \div\}$, while only 1 is provided as terminal. Protected division is used, returning 1.0 whenever the denominator is 0. Similarly to the symbolic regression problem the arity is defined by the regulatory network. A reduced function set is also used, which is composed of only $\{+, -\}$. As mentioned before, this limited function set only allows the regression of linear functions, and so this would be impossible to solve with traditional GP methodologies.

The evolutionary process will search for a solution that solves correctly the first 10 elements of the sequence. The final solution will then be tested for generalisation over the first 100 elements of the sequence.

5

Empirical Validation

The benchmark problems presented in the previous chapter provide the substrate in which one can put the computational model described in Chapter 3 and its variants to the test. Having described the tools and the tasks to solve, this chapter provides a detailed configuration of the experiences that were designed to assess the model's performance in the different classes of problems.

Section 5.1 describes the Evolution Strategy (ES) and parameter configurations common to every experiment. The basic ReNCoDe was used to explore some design alternatives of the underlying ARN model (Section 5.2), as well as to investigate the performance of the variation operators (Section 5.3). Using the best configuration resultant from the previous experiments, the ReNCoDe variants - with feedback, and with multiple outputs - were tested in the respective problem classes (Section 5.4).

Finally, in order to properly compare the results it is necessary to verify the statistical significance of the differences found across the experiments. Non-parametric statistical tests were used for this purpose (Section 5.5).

5.1 Evolution Strategy

A standard $ES(10 + 100)$ was used for all the experiments, determined by preliminary experimentation with the model. This is a steady-state approach, and a greedy selection scheme was used, i.e., every generation all the population (parents and offsprings) is competing and the 10 most fit are selected. The parents reproduce asexually, each breeding

ten offsprings, and variation is produced with the bit-flip mutation operator, at the typical rate of 1%. A run of the ES is completed when a correct solution is found or the maximum number of evaluations is reached, which was set to 10^6. Finally, in random-based methodologies the results may show great variance and thus it is important to have an appropriate number of repetitions (100 runs was the choice in this case). These parameters are summarised in Table 5.1.

Parameter	Value
ES type	steady-state
Initial Population	100
Selection	greedy
Mutation Operator	bit-flip
Mutation Rate	0.01
Max. Evaluations	10^6
Number of Runs	100

Table 5.1: Evolution Strategy parameters used across the experiments.

5.2 ARN Design Alternatives

The experiments described in this section are aimed at testing some design alternatives of the underlying ARN, providing the best configuration to use with the original proposal of ReNCoDE. The networks are produced as described in Section 3.1, and the problems used were the polinomial regression, the harmonic regression, the artificial ant, and the inverted pendulum (described in Section 4.1).

As the networks are never simulated, there are few parameters affecting ReNCoDe. Duplication and mutation (DM) is known to generate scale-free and small-world networks, similar to those found in bacteria genomes [Kuo et al., 2006]. Nonetheless, this is not necessarily beneficial in the context of optimisation as shown in [Nicolau et al., 2010] by comparing against randomly generated genomes. This study of randomly initialised genomes versus DM ones was also performed in the context of ReNCoDe. Another aspect is the genome size, which in turn influences the number of coded proteins. Different lengths were tested to find out how it affects the performance of the approach. More-

over, genes may overlap or not, changing the number of coded proteins, the organisation of the genome, and also the mutational landscape. Finally, both the threshold parameter used to trim the ARN edges (by removing every edge witch a matching strength below that value) and the DM-event mutation rate were not subject of testing, but were chosen based on the topology experiments reported by [Kuo et al., 2006, Nicolau and Schoenauer, 2009].

The summary of the values for each design detail is presented in Table 5.2. This sums a total of 20 experiments for each problem. However, the experiments with genome size 1024 bits and without overlapping genes were discarded, resulting in a total of 18 experiments. The reason behind this removal is that the total coding area of a gene with promoter and binding sites is 256 bits, making it hard for a genome with only 1024 bits to code for any proteins.

Table 5.2: Values of the different design alternatives. Initialisation may be random (rnd) or use DM events (dm). Genes may overlap or not (True/False). The size of the genome varies from 1024 to 16364 bits, with the corresponding number of DM events.

Parameter	Value
Problems	{*Polynomial*, *Harmonic*, *Ant*, *Pendulum*}
Initialisation	{*dm*, *rnd*}
Size {*#nbits*(*#DM*)}	{1024(5), 2048(6), 4096(7), 8192(8), 16364(9)}
Overlapping genes	{*True*, *False*}
DM Mutation Rate	0.02
Protein Bind Threshold	16

5.3 Genetic Operators

In Section 3.4 two types of operators are introduced - *bitwise* and genewise. The same categorisation was applied relatively to typical crossover operators used in EC. The experiments described in this section test the different operators in various classes of problems, with the basic parameterisation found in previous experiments (Table 5.3).

Three *bitwise* operators were developed to manipulate the length of the genome. The *delete* operator enables the reduction of the genome's length, and is to be used in

Table 5.3: Base parameterisation of the experiments with the genetic operators. The set of problems is also presented.

Parameter	Values
Problems	{*Polynomial*, *Harmonic*, *Ant*, *Pendulum*}
Initialisation	*rnd*
Size	4096
Overlapping genes	*False*
Protein Bind Threshold	16

conjunction with either the *transposon-like* or the *junk* operators which increase the length of the genome. These will be tested in the different domains with different length's of the section to be manipulated, from 50 bits to 400 bits. The *genewise* alternatives include two operators that operate in a similar fashion to the *bitwise*: one is able to copy complete genes, the other has the capability of removing them. Since these operate on complete genes the size parameterisation is not applicable. Moreover, for each type of operators combination, different rates of application were tested, always summing a total of 0.5, so that half the times no operator is used.

The summary of these experiments is presented in Table 5.4. The results obtained in these experiments will be compared against a control group without operators, using the same initialisation procedure and size (detailed in the previous section).

Table 5.4: Summary of the experiments with the asexual genetic operators. The different combinations of operators are presented, as well as the rates and different lengths if applicable.

Parameter	Length	Rate
transposon & delete	{50, 100, 200, 400}	{0.1/0.4, 0.2/0.3, 0.3/0.2, 0.4/0.1}
junk & delete		
gene-copy & delete	–	

5.3.1 Crossover

Concerning the crossover operators there are two variants to test - bitwise and genewise - for each of the operators. The bitwise correspond to the traditional crossover opera-

tors, while the genewise consider only non-coding regions as cutting points. The rate for crossover application is also tested, as summarised in Table 5.5.

Table 5.5: Summary of the experiments with the crossover operators. The different combinations of operators are presented, as well as the different modalities if applicable.

Parameter	Mode	Rate
one-point crossover two-point crossover uniform crossover	{bitwise, genewise}	$\{0.1, 0.3, 0.5, 0.7, 0.9\}$

5.4 Validation of the Feedback Variant

To validate the feedback variant of the original proposal, the n-bit parity, the Fibonacci sequence, the squares sequence, and the modified factorial problems(Section 4.2) were addressed.

The best performing parameterisation obtained from the experiments described in the previous section was used, summarised in Table 5.6.

Table 5.6: Parameterisation of the experiments with the feedback variant of ReNCoDe, obtained from previous experiments.

Parameter	Values
Initialisation	*rnd*
Size	4096
Overlapping genes	*False*
Protein Bind Threshold	16

The modified factorial problem (detailed in Section 4.2.4) was developed with the specific target of evaluating the scalability of ReNCoDe. It has the particularity of being tuneable by using different steps . Three experiments were designed with this problem, keeping the growth factor constant ($k = 2$), while the step of the sequence was varied from 1 to 3 in order to investigate the scalability of the method, using the reduced function set composed of $\{+, -\}$ (Table 5.7).

Table 5.7: Summary of the experiments aimed at assessing the scalability of the ReNCoDe proposal. The growth factor is constant ($k = 2$); the reduced function set is $\{+, -\}$

Parameter	Value
Step (s)	$\{1, 2, 3\}$
Function Set	$\{+, -\}$

5.5 Statistical Validation.

In order to compare the different experiments it is necessary to perform a statistical analysis of the results. Since the samples do not follow a normal distribution the analysis was performed using non-parametric tests. Moreover, since there are more than two groups to analyse and different populations were used, the appropriate statistical test is the Kruskal-Wallis test.

This test was used at a significance level of 5%. For the significant differences, pairwise *post-hoc* comparisons were made using the Mann-Whitney-Wilcoxon test [R Development Core Team, 2008]. Particularly, one-tailed tests were performed in both directions (greater and lesser). This means that for each pair two tests were performed, in order to be able to assess if there is a statistically significant difference, either positive or negative. Since we are dealing with multiple pairwise comparisons, Bonferroni correction was applied [Field, 2003].

6

Results and Discussion

Having described the experimental setup for the different classes of problems in the last chapter, here the results obtained from those experiments will be analysed and discussed. The original ReNCoDe is discussed in Section 6.1, followed by the use of variation operators besides bit-flip mutation in Section 6.2. Finally, the variant with feedback connections is analysed in Section 6.3.

6.1 ARN Design Alternatives

In Section 5.2 a set of experiments was presented whose goal is to determine what is the best configuration for underlying ARN model in ReNCoDe, using the basic model described in Section 3.2.

In this section we present the results obtained from those experiments. Using the problems from Section 4.1, three parameters were investigated: i) initialisation by DM-events (*dm*) opposed to random initialisation (*rnd*), ii) the size of the genome (indicated by the number of DM events from 5 to 9), and iii) whether genes overlap or not (indicated by a *T* or *F* respectively).

The algorithm was effective in finding optimal solutions for all the problems, as shown by the success rates displayed in Figure 6.1. With exception of the extreme size cases (5/9 DM events), the algorithm finds a solution to the problems in most of the runs, especially when random initialisation is used. The evidence that randomly initialised genomes perform better is more clear when one looks at the distribution of the number of evaluations necessary to find an optimal solution (Figure 6.2). Despite the high number of

outliers, one can see that runs with randomly initialised genomes of moderate size (7/8 DM events, 4096/8192 bits) tend to find an optimal solution with less effort and more often. Examples of the programs extracted for each of the problems are presented in Figure 6.3, with exception of the controller for the inverted pendulum which can be seen in Figure A.3. To see the detailed summary of the results for each experiment and problem the reader may consult Table A.1.

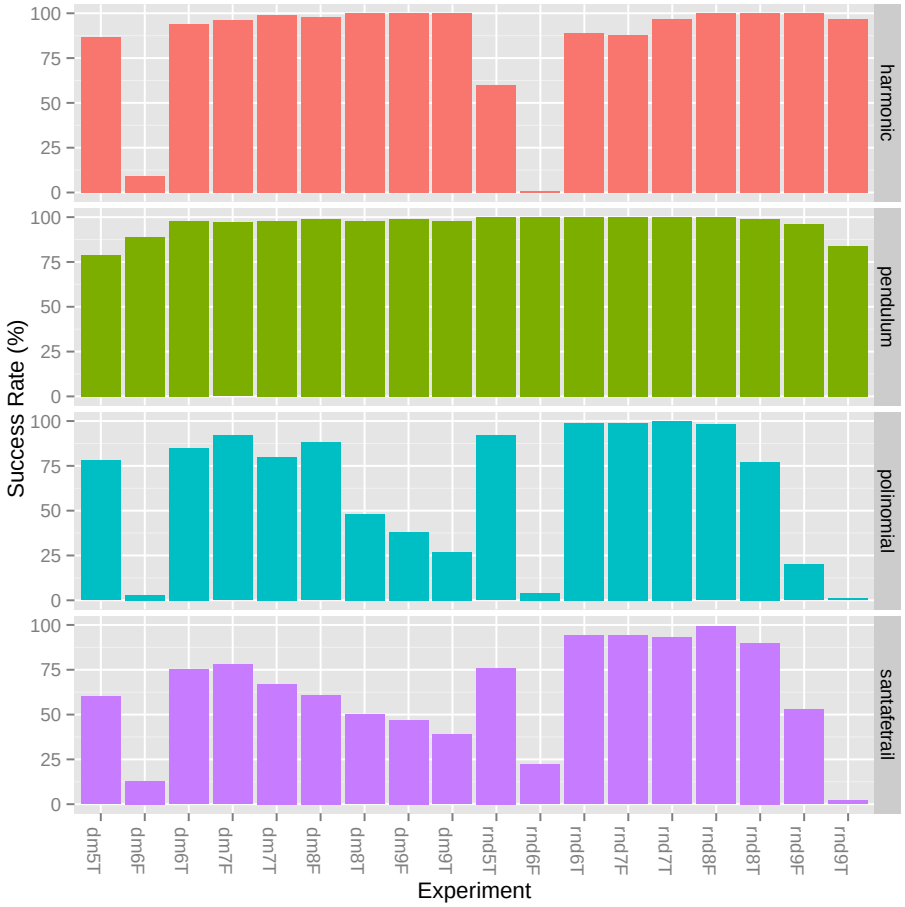


Figure 6.1: Success rates for ReNCoDe in typical benchmarks. The results are split by problem and the labels refer the initialisation method (*dm* or *rnd*), the size (using the corresponding DM event number from 5 to 9), and if genes overlap or not (*T* or *F*). With exception of the extreme cases (where dm = 5 or dm = 9) the algorithm finds a solution to the problems in most of the runs, especially when random initialisation is used.

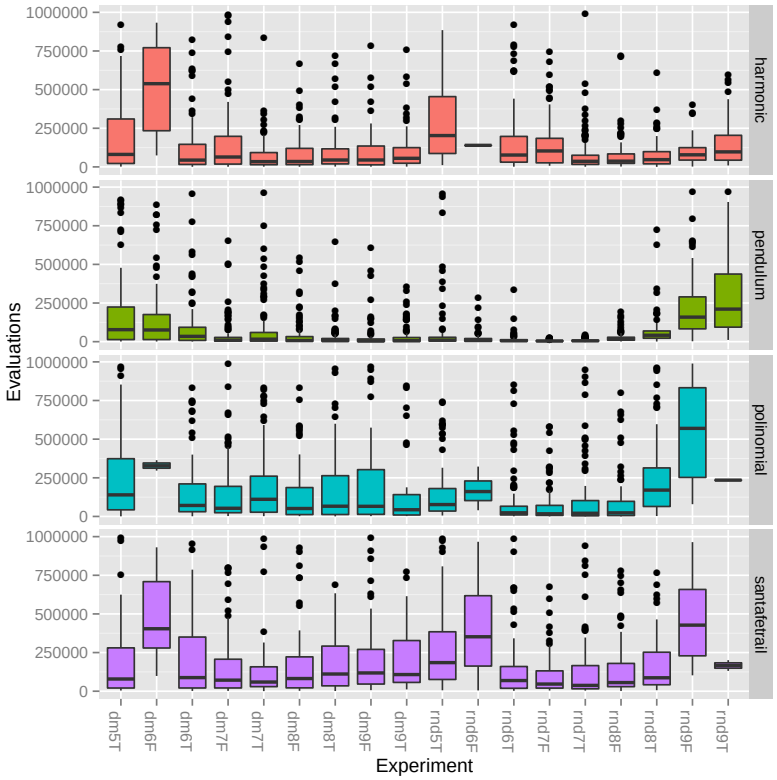
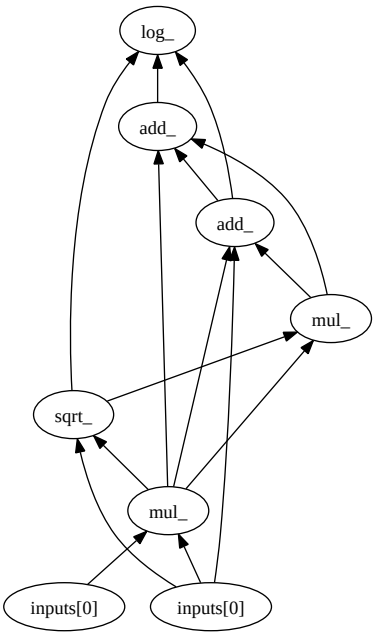
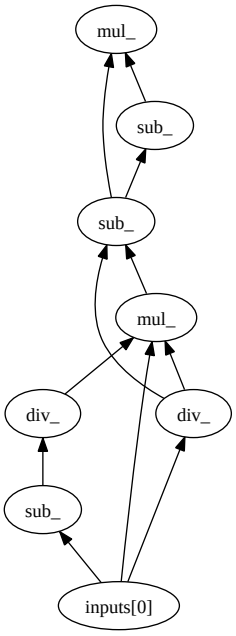


Figure 6.2: Distribution of the number of evaluations necessary to find a solution using ReNCoDe in typical benchmarks. The results are split by problem and the labels refer the initialisation method (*dm* or *rnd*), the size (using the corresponding DM event number from 5 to 9), and if genes overlap or not (*T* or *F*). The algorithm is more efficient using random initialisation and moderate genome sizes.

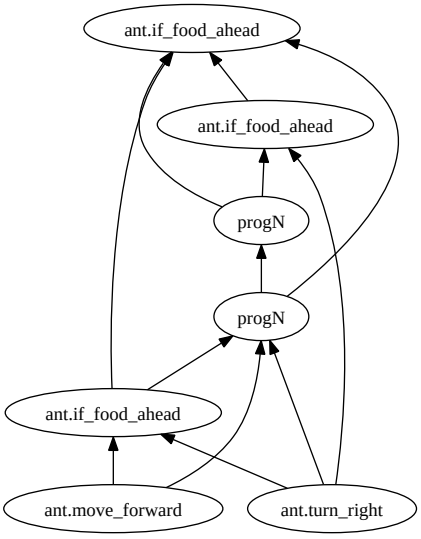
The overlap observed amongst the experiments in Figure 6.2 and the high number of outliers indicate that the differences amongst the means may not be statistically significant. In order to confirm the data was split by problem and the statistical tests were performed as described in Section 5.5. The results from the Kruskal-Wallis test indicate that there is significant difference amongst the populations in every problem (Table 6.1). However, the results of the pairwise comparisons shown in Figure 6.4 and 6.5 indicate that most of the differences are not statistically significant (for the polinomial and santa fe trail problems see Figures A.1 and A.2). In these figures the tiles are marked with *less* if the number of evaluations of the experiment in the y axis is significantly less than the one on the x axis, with *greater* otherwise, or *not significant* if the difference is not statistically significant (the



(a) Harmonic Regression



(b) Polinomial



(c) Santa Fe Trail

Figure 6.3: Examples of the programs evolved for each of the standard problems. Due to its size, the controller for the inverted pendulum is presented in Figure A.3.

detailed results can be found in Listings A.1-A.8). Although there are some configurations that consistently perform worse (where the difference is significantly greater), as is the case of random initialisation with size 16Kbits (*dm9T,dm9F,rnd9T,rnd9F*), there is not any configuration that is best across all domains. However the tendency is that runs with randomly initialised genomes need less evaluations to find a solution (see the cases of the configurations from *rnd7F* to *rnd8F*).

Table 6.1: Kruskal-Wallis test for each problem using ReNCoDe.

Problem	chi-squared	df	p-value
Harmonic	129.5448	17	$< 2.2 \times 10^{-16}$
Pendulum	596.3981	17	$< 2.2 \times 10^{-16}$
Polinomial	174.6874	17	$< 2.2 \times 10^{-16}$
SantaFeTrail	161.0365	17	$< 2.2 \times 10^{-16}$

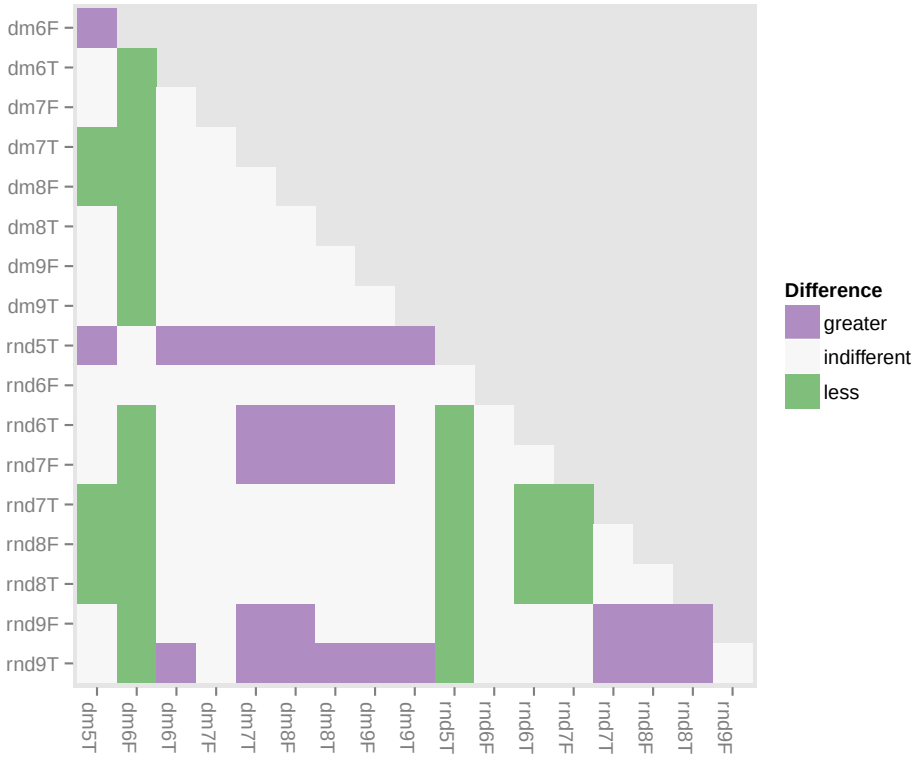


Figure 6.4: Mann-Whitney-Wilcoxon test results for the design alternatives with the harmonic regression problem, showing *less* if the number of evaluations of the experiment in the *y* axis is significantly less than the one on the *x* axis, with *greater* otherwise, or *not significant* if the difference is not statistically significant.

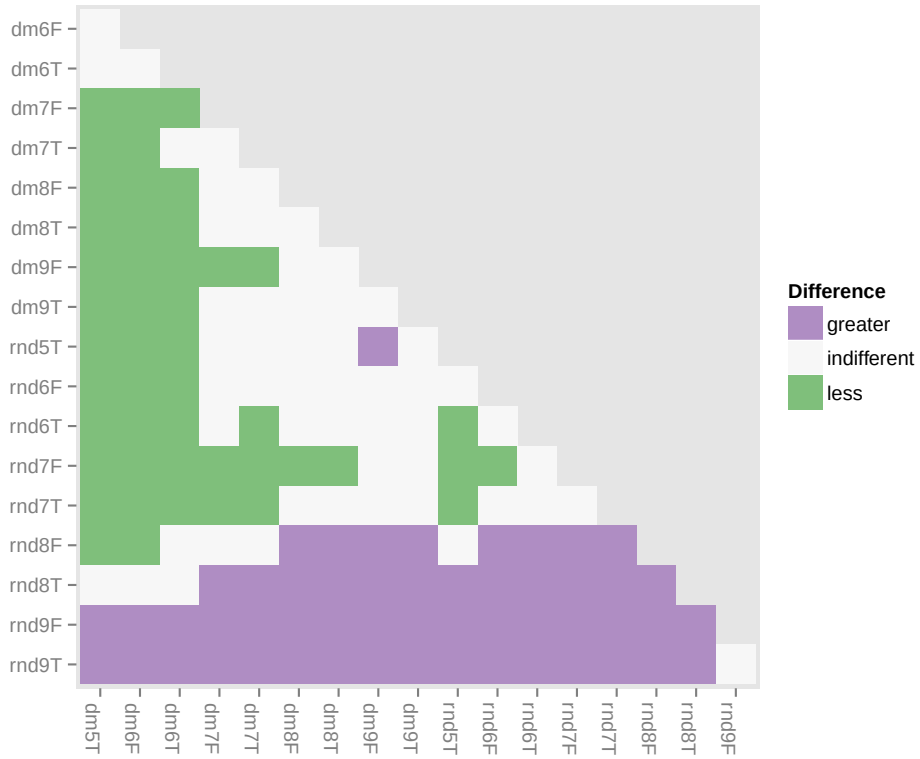


Figure 6.5: Mann-Whitney-Wilcoxon test results for the design alternatives with the inverted pendulum problem, showing *less* if the number of evaluations of the experiment in the *y* axis is significantly less than the one on the *x* axis, with *greater* otherwise, or *not significant* if the difference is not statistically significant.

Generalisation

Two of the problems presented in Section 4.1 have a generalisation task. First, in the harmonic curve regression the best solution found in each run is tested with an extended set of points. Second,in the inverted pendulum problem the best controller found in each run is tested with a set of 625 initial positions that cover the full range of values of the control variables. Statistical tests were performed to compare the parameterisations in each problem (the detailed results of which are presented in Listings A.9-A.12). The complete numerical results on the generalisation tasks can be found in Table A.2.

In the case of the harmonic curve regression one is interested in solutions whose generalisation error is minimum. Figure 6.6 shows the mean fitness obtained in the gen-

Table 6.2: Kruskal-Wallis test for the generalisation tasks of the harmonic curve regression and the inverted pendulum problems.

Problem	chi-squared	df	p-value
Harmonic	82.3739	17	1.448×10^{-10}
Pendulum	236.7037	17	$< 2.2 \times 10^{-16}$

eralisation task plotted with the minimum and maximum values found. The fitness is measured by the normalised root mean squared error, in percentage. Performing the statistical analysis (presented in Table 6.2 and Figure 6.7) one can see that randomly initialised genomes perform better (lower error), with statistically significant difference against de DM initialisation in the case of genomes with moderate size (see the cases from *rnd6T* to *rnd7T*).

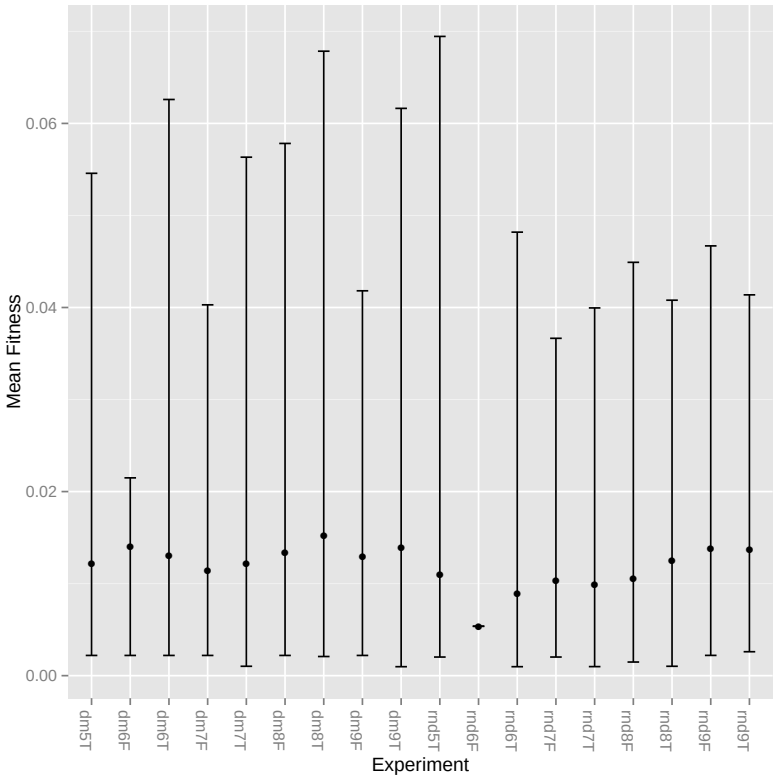


Figure 6.6: Results for the generalisation task of the harmonic curve regression problem. The mean fitness approximating the function is presented with standard error. The fitness corresponds to the normalised root mean square error (in percentual points).

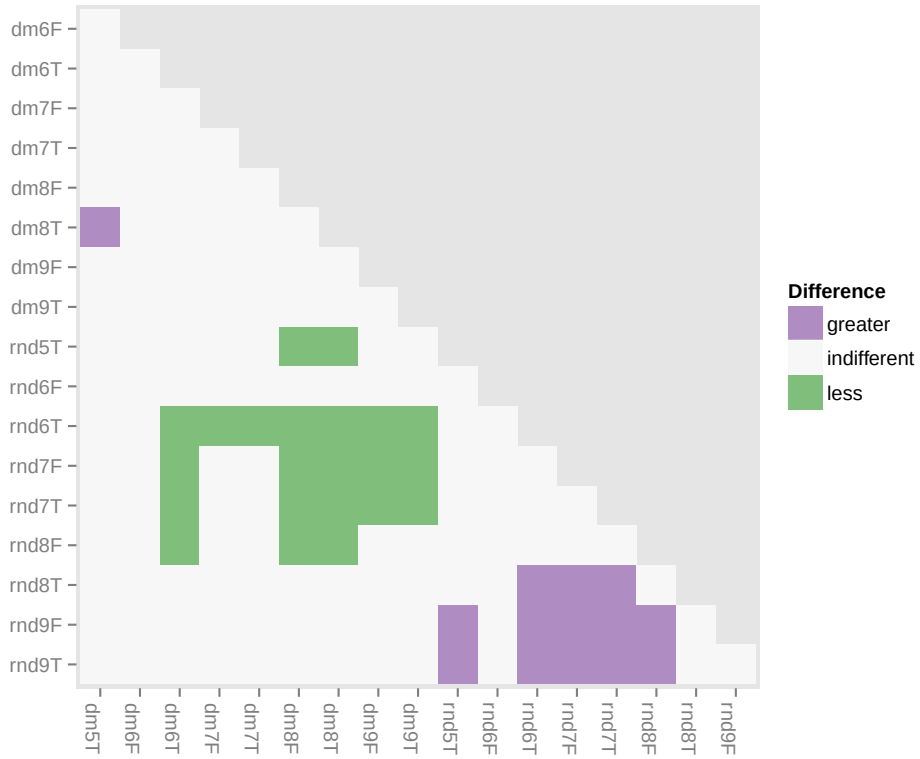


Figure 6.7: Mann-Whitney-Wilcoxon test results for the generalisation task of the harmonic curve regression problem, showing *less* if the number of evaluations of the experiment in the *y* axis is significantly less than the one on the *x* axis, with *greater* otherwise, or *not significant* if the difference is not statistically significant.

In the inverted pendulum scenario we are interested in the solutions with the highest generalisation fitness, which in this case corresponds to the number of successful simulations out of 625. In Figure A.4, that shows the mean fitness obtained in the generalisation task plotted with the minimum and maximum values found, one can see that DM generated genomes tend to generalise better. Moreover, the mean is very low compared to the maximum value, which indicates that most of the solutions evolved generalise poorly over the complete range of the input variables. The statistical pairwise tests results (displayed in Figure A.5) indicate that there is not a configuration that is better than the others, although one can conclude that randomly initialised genomes with extreme sizes (either small or large) generalise worst than the remaining configurations.

The best controller successfully kept the pole balanced in 495 out of the 625 initial

positions. The summary of the tests is presented in Figure 6.8. One can see that the controller only fails on extreme positions (when the input variables are close to either the maximum or the minimum), and that it is consistent since the displayed graph is symmetric.

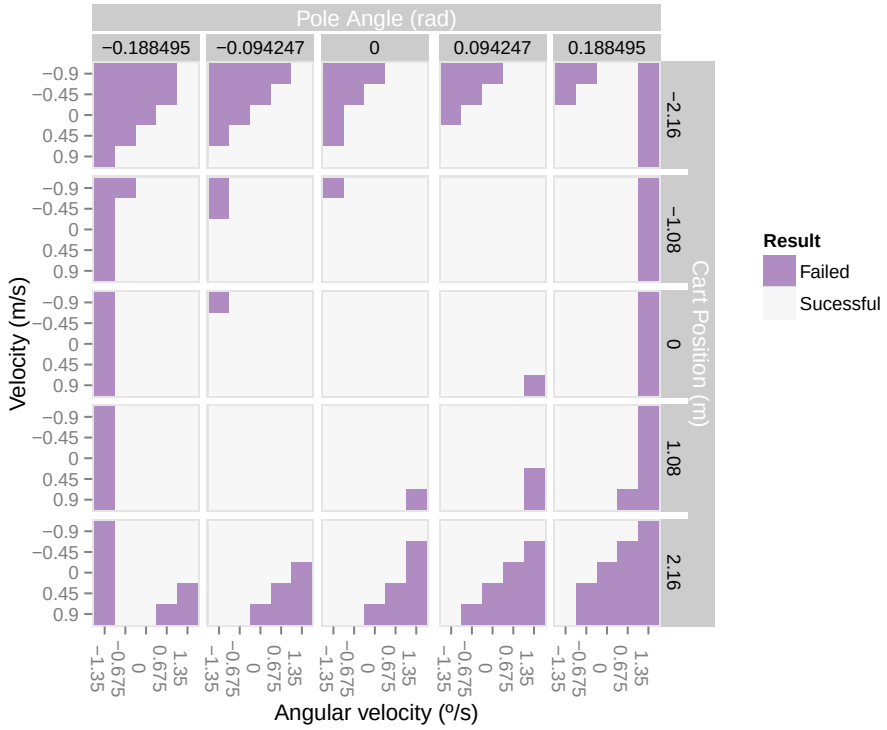


Figure 6.8: Summary of the results of the generalisation tests for the best controller of the inverted pendulum, which was successful in 495 of the 625 initial states. All the combinations of the four input variables used in the generalisation test are displayed in the matrix: the cart position (x), the pole angle (θ), the velocity ($\dot{x}$), and the angular velocity ($\dot{\theta}$)

6.2 Genetic Operators

With the insights obtained in the previous section about the underlying ARN model parameterisation, the experiments to determine if the introduction of the operators is beneficial (and under what conditions) were performed (see Section 5.3). For that purpose the baseline configuration chosen from the results presented in the previous section was *rnd7F*, meaning that genomes were randomly initialised with 4096 bits, and without overlapping genes. Both the bitwise and genewise asexual operators were investigated.

As mentioned in Section 3.4, two bitwise operators were used (*junk*, and *transposon*) coupled with a *delete* operator. These pairs will be referred to as *JD* and *TD*. In the case of the gene wise operators there is only one combination of the *genecopy* and *genedelete* operators, which will be identified by *GCD*. The rates configuration indicates the rate used for each option in a pair of operators, with the *delete* always in the last position. In the tables and graphics they are shortened to the digit behind the decimal point. For instance, *TD*.32.100 indicates that the *transposon-delete* pair was used with 0.3 and 0.2 as the respective rates, and with section length of 100 bits.

Let us analyse the results amongst the operators only. The complete numerical results can be consulted in Table A.3. In terms of success rates (Figure 6.9), although there is not much variation there is some tendency to reach more often an optimum using the rates configurations *0.3-0.2* and *0.4-0.1*. Such a conclusion is not possible relative to the operator length (there is not a tendency generalisable to all the problems/operators). It also noticeable that in the symbolic regression problems the *junk* operator does not perform as well as the others, as there is a visible decrease in the success rates.

Concerning the effort (the number of evaluations before termination), the analysis for the harmonic regression is presented in Figure 6.10, while for the remaining problems is in Figures A.6-A.11. The Kruskal-Wallis test results are displayed in Table 6.3, indicating the presence of statistically significant differences. One can conclude that using the rates parameterisations *0.3-0.2* and *0.4-0.1* the operators perform better. However, only in few cases there is statistical significance, see for instance *TD.41.400* in the harmonic regression problem (Fig. 6.11). Moreover, one confirms that in the symbolic regression domain both the *transposon* and the *genewise* operators perform better than the *junk* (rarely with statistical significance, see the case of *JD.14.200*). In the pendulum and Santa Fe trail such a generalisation is not possible though.

Table 6.3: Kruskal-Wallis test for the asexual operators parameterisation in each problem.

Problem	chi-squared	df	p-value
Harmonic	98.6959	44	4.461×10^{-6}
Pendulum	62.3817	44	3.535×10^{-2}
Polynomial	95.608	44	1.095×10^{-5}
SantaFeTrail	72.0114	44	4.869×10^{-3}

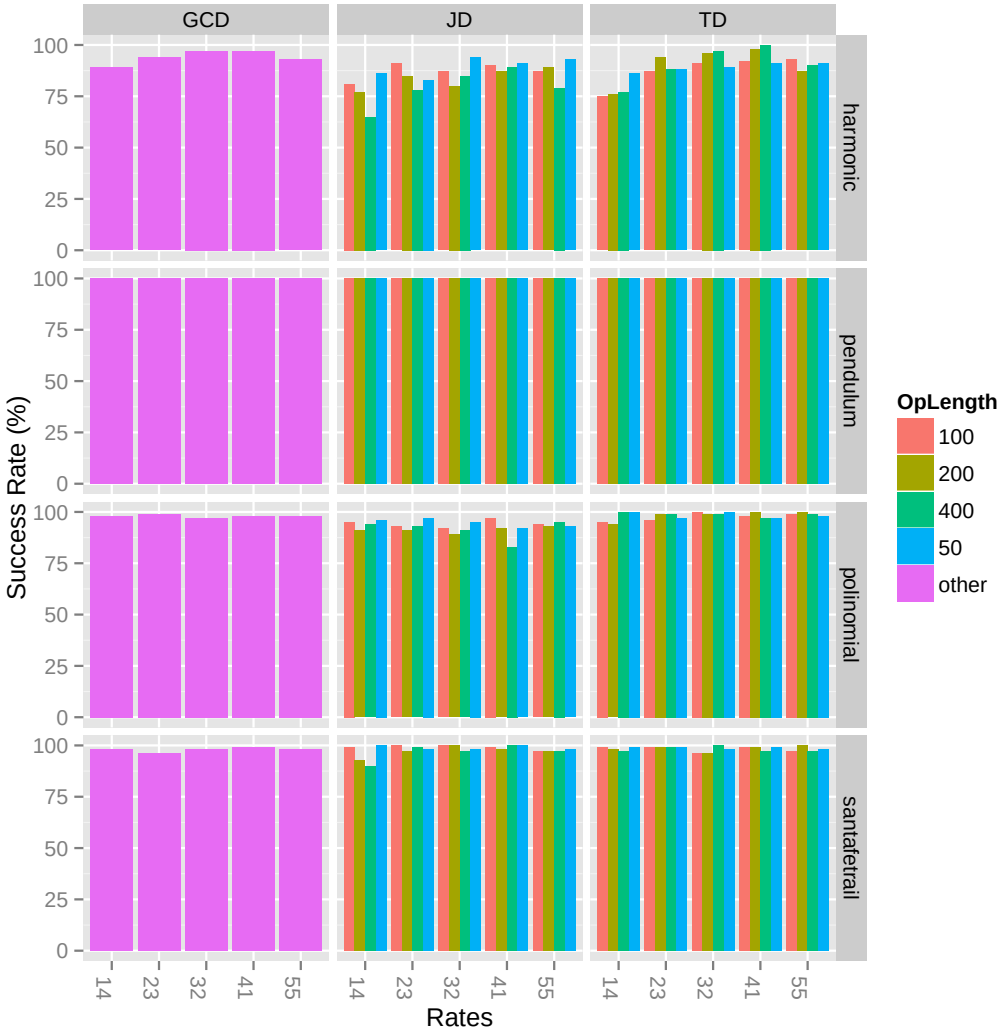


Figure 6.9: Success rates for the experiments on the ReNCoDe asexual operators (both bitwise and genewise operators). The results are split by problem and operator, where *JD, TD*, and *GCD* correspond respectively to the bitwise *junk* and *tranposon* (coupled with the *delete* operator), and to the *genecopy* and *genedelete* operators. The rates are represented by the first decimal place, that is, *14* stands for *0.1-0.4*, the order consistent with the operators labels.

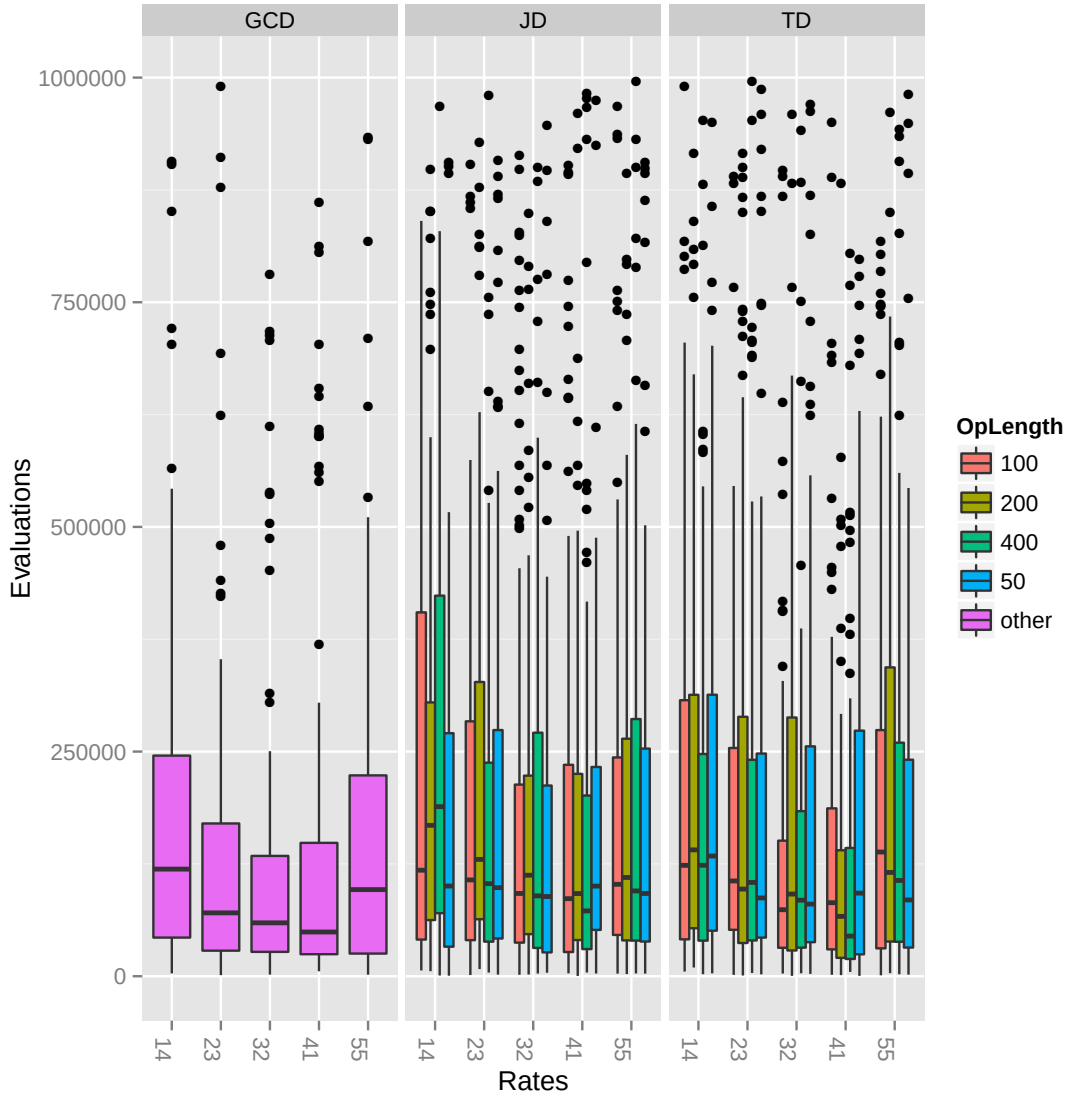


Figure 6.10: Distribution of the number of evaluations necessary to find an optimal solution for the harmonic regression problem, using the asexual operators. The results are split by operator, where *JD, TD*, and *GCD* correspond respectively to the bitwise *junk* and *tranposon* (coupled with the *delete* operator), and to the *genecopy* and *genedelete* operators. The rates are represented by the first decimal place, that is, *14* stands for *0.1-0.4*, the order consistent with the operators labels.

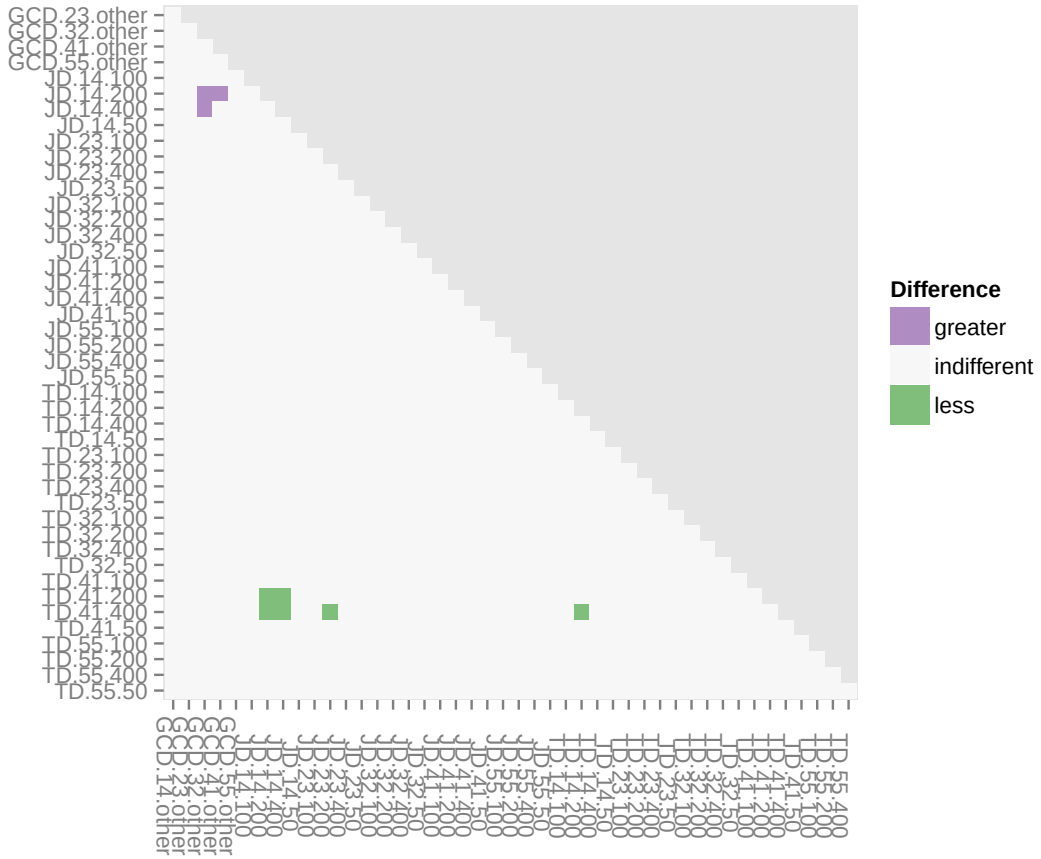


Figure 6.11: Pairwise statistical tests results for the number of evaluations necessary to find an optimal solution for the harmonic regression problem, using the asexual operators. In the labels *JD, TD*, and *GCD* correspond respectively to the bitwise *junk* and *tranposon* (coupled with the *delete* operator), and to the *genecopy* and *genedelete* operators. The rates are represented by the first decimal place, that is, *14* stands for *0.1-0.4*, the order consistent with the operators labels. The section length is appended at the end.

Finally, the results were compared with the baseline parameterisation chosen from the experiments discussed in the previous section (*rnd7F*). For this effect the results were split by operator and every configuration was used in the contrasts. The summary of the statistical comparisons is shown in Figure A.12, indicating that only one situation (harmonic regression, using the transposon and delete operators at 0.4-0.1 rates respectively and size 400 bits) performs better with statistical significance.

6.2.1 Crossover

The application of typical crossover operators (presented in Section 3.4) was also investigated using different rates, as described in Section 5.3.1. Both *bitwise* and *genewise* versions of the 1-point, 2-point, and uniform crossover operators were implemented. As mentioned before, the *bitwise* version is blind to the genome structure, while the *gene wise* operators are aware of the start and end bits of the genes.

The analysis was performed as with the previously discussed operators. First, comparisons were drawn amongst the operators, and then the operators were compared to the baseline experiment (*rnd7F*, see Section 6.1). The complete numerical results can be found in Table A.4.

The effectiveness of the crossover operators can be observed in Figure 6.12. While there is no distinguishable difference between most of the operators at any of the rates, the genewise uniform crossover is clearly less effective, reaching optimal solutions in fewer experiments than the remaining.

The efficiency of the operators presents a bigger variation amongst the different rates of application. As one can see by the distribution of the number of evaluations presented in Figure 6.13 (for the harmonic regression problem), there is a tendency that points out the 0.5 rate as the appropriate choice amongst the different types of operators[1]. However, the statistical tests indicate that there is no statistical significant difference amongst the experiments, as shown by the results of the Kruskal-Wallis test presented in Table 6.4. Moreover, although we can reject the null hypothesis in the remaining cases (there is significant difference amongst the populations), the post-hoc tests[2] indicate that only a reduced number of differences are significant. The few cases where there is a statistical significant difference, allow to conclude only that the uniform genewise crossover performs worse than the remaining.

Finally, the results were compared with the baseline parameterisation chosen from the experiments discussed in the previous section (*rnd7F*). For this effect the results were split by crossover operator and every rate configuration was used in the contrasts. The results of the statistical comparisons indicated that none of the operators performed better than the baseline with statistical significance.

[1] The box-plots for the remaining problems can be found in the Appendices, Fig. A.13, A.15, and A.17

[2] The results of the *post-hoc* comparisons for each of the remaining problems are presented in the Appendices, Fig. A.14, A.16, and A.18

Table 6.4: Kruskal-Wallis test for the crossover operators parameterisation in each problem.

Problem	chi-squared	df	p-value
Harmonic	31.3092	29	0.351
Pendulum	64.5194	29	1.639×10^{-4}
Polynomial	48.1773	29	1.409×10^{-2}
SantaFeTrail	49.4243	29	1.041×10^{-2}

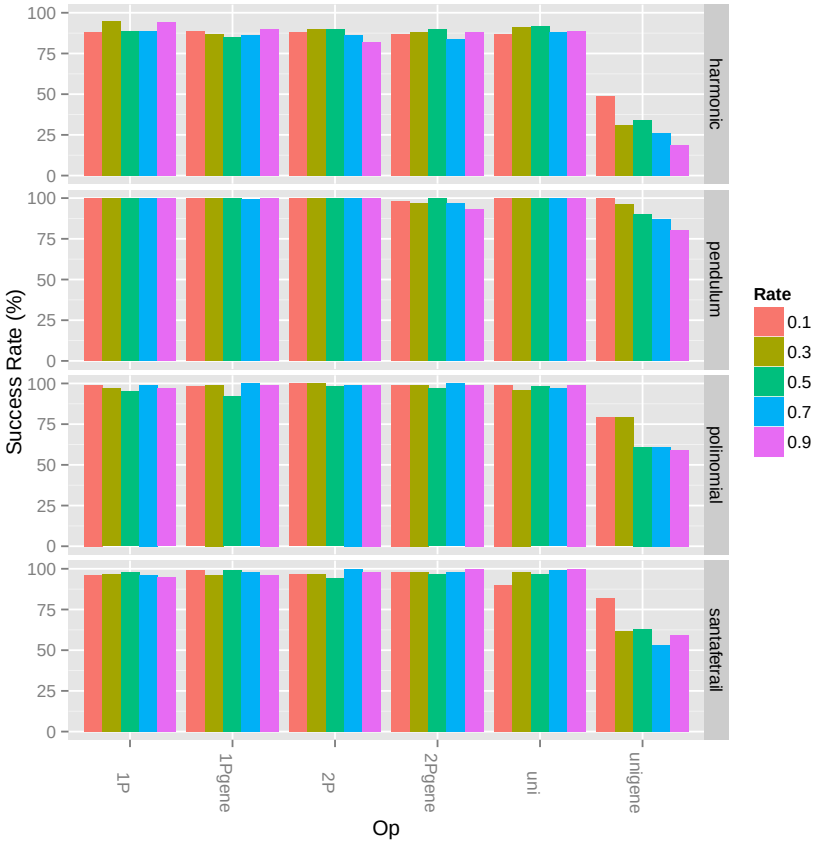


Figure 6.12: Success rates for the experiments on the ReNCoDe crossover operators (both bitwise and genewise). The results are split by operator, where *1P*, *2P*, and *uni* correspond respectively to the bitwise *1-point*, *2-point*, and *uniform* crossover operators, with the suffix *gene* applied to the genewise versions.

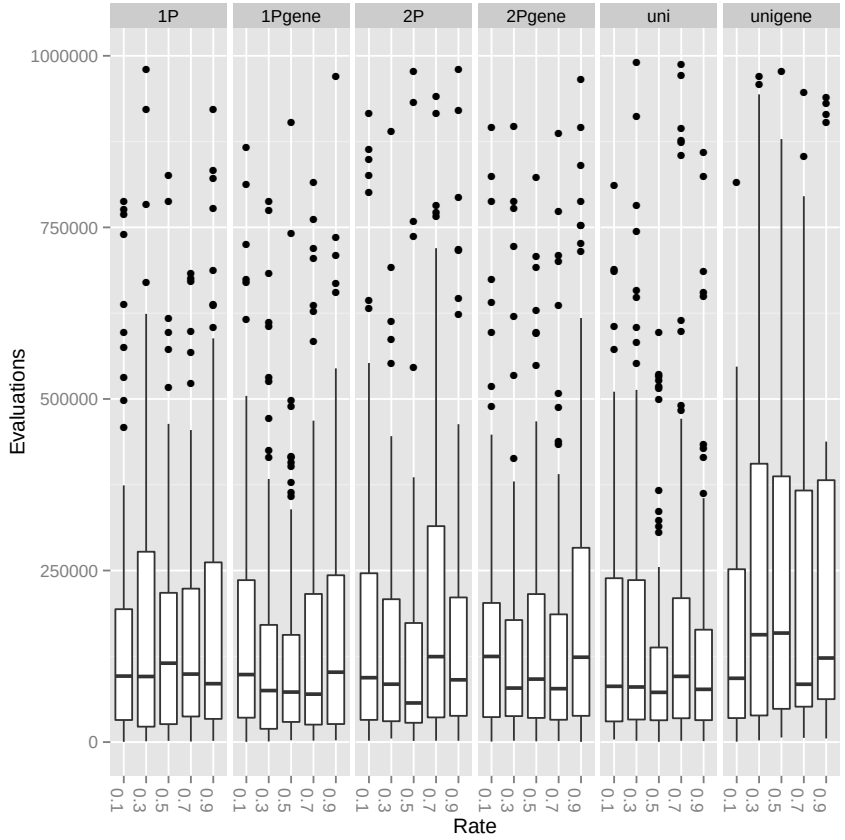


Figure 6.13: Distribution of the number of evaluations necessary to find an optimal solution for the harmonic regression problem, using the crossover operators. The results are split by problem and operator, where *1P*, *2P*, and *uni* correspond respectively to the bitwise *1-point*, *2-point*, and *uniform* crossover operators, with the suffix *gene* applied to the genewise versions.

6.3 Validation of the Feedback Variant

To validate the applicability of the feedback variant of the original proposal to the different problems presented in Section 4.2, the best performing parameterisation obtained from the experiments in the previous section was used (Table 6.5).

The n-bit parity, the Fibonacci sequence, the squares sequence, and the modified factorial (Section 4.2) have in common that the fitness cases used during the evolutionary runs are a small subset of those used to assess the generalisation abilities of the evolved solutions.

The task is to extrapolate some function from a reduced set of known cases (either

Table 6.5: Parameterisation of the experiments with the feedback variant of ReNCoDe, obtained from previous experiments.

Parameter	Values
Initialisation	*rnd*
Size	4096
Overlapping genes	*False*
Protein Bind Threshold	16

the first terms of a sequence or the results for some points of a function). The subset of the fitness cases used during the evolutionary run is usually an easy task, with most runs taking only few thousand evaluations to reach optimal individuals and a success rate close to 100% in every problem (Table 6.6). Actually, in all of the problems there were runs in which a fit individual was found on the initial - randomly generated - population (MinEval). This demonstrates the adequacy of the representation to map artificial gene regulatory networks into program graphs in these problem domains.

Table 6.6: Summary of the results for the problems with recursion: n-bit parity, the squares sequence, the fibonacci sequence, and the modified factorial ($k = 2, s = \{1, 2, 3\}$). The success rate, the average, standard deviation, and the minimum number of evaluations, as well as the average and minimum number of functions are presented, respectively. The generalisation rate in the last column is relative to the successful runs only.

Problem	S.R.	AvgEval	StdDev	MinEval	AvgFun	MinFun	G.R.
nbitparity	100	10871	34290	100	5.4	4	91
squares	100	16478	45725	100	4.8	4	99
fibonacci	100	37009	65622	100	5.4	4	100
modfactorial-k2s1	100	3087	3890	100	5.3	4	78
modfactorial-k2s2	100	22065	34791	100	5.1	4	100
modfactorial-k2s3	5	567500	274475	220600	7.4	7	100

The exception was the modified factorial problem, showing that there are scalability issues. The success rate achieved (percentage of runs that solved the problem for the first ten elements) decreased drastically when $s = 3$. The results of the first two levels of the step are in line with those obtained in the squares sequence (similar to $s = 1$) or the Fibonacci sequence (similar to $s = 2$), but finding solutions that are able to keep track of

the values from three iterations earlier was more difficult.

An interesting property of the evolved solutions is the reduced number of functions in the programs (AvgFun and MinFun). Although there is evidence of bloat even in the graphs where the number of functions was minimal, it is negligible when compared to typical GP. One can see particular cases of minimal programs with bloat in the next section, where the generality of exemplar solutions for each problem is discussed.

Generality

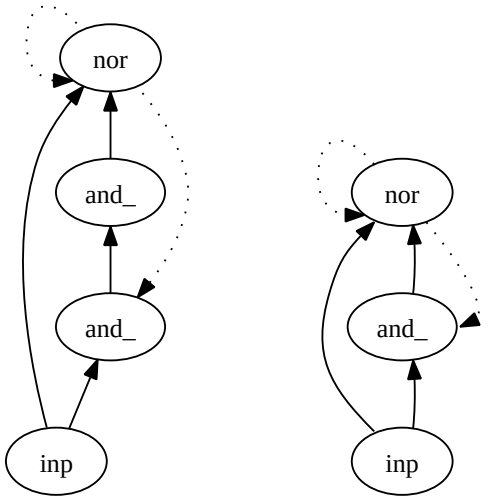
Most of the evolved solutions were able to solve correctly the generalisation task (Table 6.6, G.R.). The reported values suggest that the failures are due to premature convergence in the easier problems, since that the quicker it is to find a correct solution (column AvgEval), the lower is the generalisation rate.

As mentioned before, given the recursive nature of the problems, experimental evaluation over the complete range of possible values is not viable because of i) the computational effort (and consequently the total computation time necessary), and ii) the limitations intrinsic to computer architectures (for instance, the limited range of the number types). Although the number of fitness cases used in generalisation is usually at least ten times greater than the size of the training set, it still is just a subset of the possible universe. The only way to determine whether the evolved solutions are general or not is to verify analytically each particular phenotype that successfully solved the experimental generalisation task. With that in mind, an example solution for each problem will be formalised in the remaining of this section.

Even n-bit parity. In this problem one wants to know whether the programs return the correct parity bit for any input size. Taking the program graph depicted in Figure 6.14 as an example one can prove that the evolved solution is correct for the entire domain of the problem. In the program graph each node represents either a gate or the binary string input. The functions get their input values from the nodes connected through the in-edges. The dotted edges represent the feedback connections (because it is a feed-forward circuit, the function was not evaluated yet and keeps the previous result similarly to a flip-flop in a digital circuit[3]). If a function has only one in-edge the neutral

[3] A flip-flop is a circuit that has two stable states, and it is the basic storage element in sequential logic. The feedback connections behave in particular like a *delay* or *data* flip-flop by capturing the value of the

element of the basic operation (*and* or *or*) is added. Before the first evaluation all the nodes hold 1 as the result. As described earlier, in nodes with more than two inputs the corresponding function is successively applied with arity two (from left to right), so that for instance $nor(a, b, c) = nor(nor(a, b), c)$ or $nand(a, b, c) = nand(nand(a, b), c)$. For the remaining operators it is irrelevant due to the associativity and commutativity properties of boolean algebra.



(a) Evolved. (b) Simplified.

Figure 6.14: One of the smallest programs evolved that generates the parity bit of any binary string: a) the evolved solution, b) a simplification by removal of a neutral node with only one input. The nodes represent gates or the input stream. The output of the program is taken from the top (the *nor* node). The binary string is streamed through the *inp* node. Functions take their inputs from the nodes where their in-edges point from. The dotted edges represent feedback connections.

Considering the simplified program, let f_n denote the output at the *nor* node, g_n the output of the *and* node, and x_n the input bit at iteration n (corresponding also to the position in the string). Due to the initialisation procedure, one has $f_{-1} = g_{-1} = 1$. Then,

$$g_n = x_n \wedge f_{n-1}$$
$$f_n = (f_{n-1} \veebar x_n) \veebar g_{n-1}$$

which leads us to the algebraic expression that defines the n-bit even parity function:

input node between each evaluation (the rising edge of the clock).

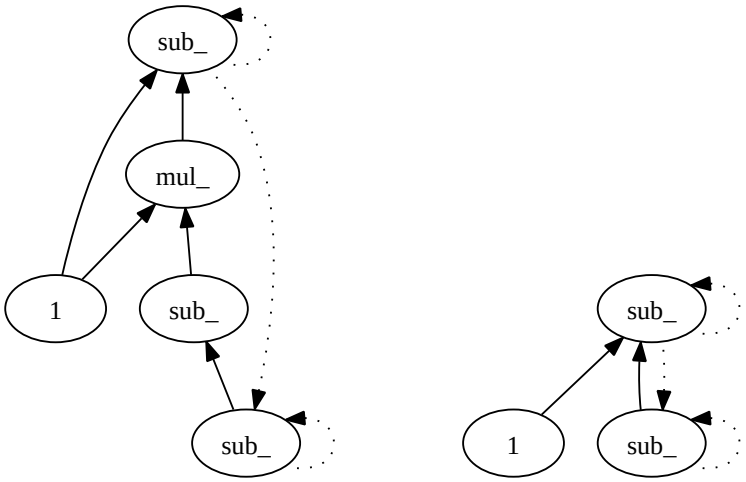
$$\begin{aligned} f_n &= (f_{n-1} \veebar x_n) \veebar (x_n \wedge f_{n-1}) \\ &= f_{n-1} \oplus x_n \\ &= 1 \oplus \bigoplus_{i=0}^{n} x_i \ , \ n = 0, 1, \ldots, \infty \end{aligned}$$

Despite the approaches being different and not directly comparable, the achievements of ReNCoDe are competitive with those presented in Section 4.2.1, showing less effort. However, as one can see from the generality demonstration presented above, and given that the circuits are iterated over the input stream, the evolutionary process just has to find a program that successively performs a 'XOR' of the current input with the previous result, actually making the task easier.

The Fibonacci sequence. This sequence has been subject of study for many mathematicians and other scientists for a long time. From the results that can be found in the literature, it is hard to evolve systems that generalise correctly, specially after the first 74 elements (although this may be due to losses of precision in the conversions from integer to floating point values, for instance when using protected division operators). The goal here is to show that the evolved programs are able to generate Fibonacci numbers sequentially beyond the first hundred elements, only limited by the hardware constraints.

Consider the program presented in Figure 6.15, one of the smallest evolved. Similarly to the binary program presented for the n-bit parity, each function takes its input values from the in-edges, and the dotted edges represent feedback connections. If a node has only one in-edge connection the respective neutral element of the operation is used. The circuit is then iterated updating the functions in a bottom-up fashion. In each iteration the result at the top node is retrieved as an element of the sequence. Every node holds the value 1 before the first evaluation, and there are not conversions from integer to floating point precision. In nodes with more than two inputs the corresponding function is successively applied with left associativity, so that for instance $sub(a, b, c) = sub(sub(a, b), c)$.

Let f_n denote the output of the program, and g_n the output of the lower subtraction node at iteration n. One can write the general expression for the sequence generated by g_n , and the sequence generated at the output node which depends of the former



(a) Evolved. (b) Simplified.

Figure 6.15: One of the smallest programs evolved that generates the Fibonacci sequence: a) the evolved solution, b) a simplification by removal of a neutral node with only one input and one multiplication by 1. The nodes represent arithmetic operators. The output of the circuit is taken from the top. Each iteration the nodes are updated from bottom to top and the result is taken as an element of the sequence. Functions take their inputs from the nodes where their in-edges come from. The dotted edges represent feedback connections.

(Equations 6.1 and 6.2), knowing that $g_{-1} = 1$ and $f_{-1} = 1$.

$$g_n = g_{n-1} - f_{n-1} = 1 - \sum_{k=0}^{n} f_{k-1} \tag{6.1}$$

$$f_n = 1 - g_n - f_{n-1} \tag{6.2}$$

Hence, replacing g_n in Eq. 6.2 holds the definition of the Fibonacci number, as shown in Equation 6.3:

$$
\begin{aligned}
f_n &= (\sum_{k=0}^{n} f_{k-1}) - f_{n-1} \\
&= (\sum_{k=0}^{n} f_{k-1}) - (\sum_{k=0}^{n-1} f_{k-1}) + f_{n-2} \\
&= f_{n-1} + f_{n-2} \qquad (6.3)
\end{aligned}
$$

The squares sequence. The goal of this problem is to find a solution that generates the function $f_n = n^2$, without using multiplication or division, as this would render the problem trivial. The functionality can still be achieved using the feedback connections to access previous results, as will be demonstrated.

Consider the program presented in Figure 6.16, one of the smallest evolved (the program graph is initialised and executed as described for the previous problem). Let f_n represent the output of the program, g_n of the addition node, and h_n of the lowest subtraction node at iteration n, one can write the corresponding symbolic expressions (Equations 6.4 to 6.6), given that $h_{-1} = g_{-1} = f_{-1} = 1$.

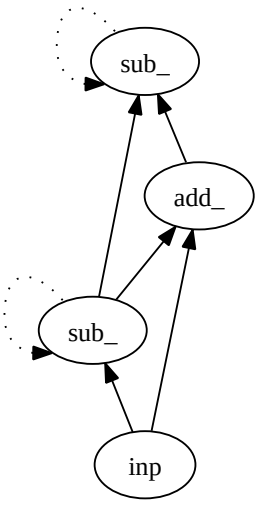


Figure 6.16: One of the smallest programs evolved that generates the squares sequence. The nodes represent arithmetic operators. The output of the circuit is taken from the top node. Each iteration the nodes are updated from bottom to top and the result is taken as an element of the sequence. Functions take their inputs from the nodes where their in-edges come from. The dotted edges represent feedback connections.

$$h_n = h_{n-1} - 1 = -\sum_{k=0}^{n} 1 = -n \tag{6.4}$$

$$g_n = h_n + 1 = 1 - n \tag{6.5}$$

$$\begin{aligned} f_n &= f_{n-1} - h_n - g_n \\ &= f_{n-1} + 2n - 1 \end{aligned} \tag{6.6}$$

If f_n is telescoped through the previous terms, it follows that:

$$\begin{aligned} f_n &= f_{n-1} + 2n - 1 \\ &= f_{n-2} + (2(n-1) + 1) + 2n - 1 \\ &= f_{n-2} + (2n - 3) + (2n - 1) \\ &= f_{n-3} + (2n - 5) + (2n - 3) + (2n - 1) \\ &= f_{n-3} + 3(2n) - (5 + 3 + 1) \\ &= f_{n-k} + k(2n) - \sum_{i=1}^{k} (2i - 1) \end{aligned} \tag{6.7}$$

Hence, using Equation 6.7, for $k = n + 1$, it follows[4]:

$$\begin{aligned} f_n &= f_{n-(n+1)} + (n+1)2n - \sum_{i=1}^{n+1} 2i - 1 \\ &= f_{-1} + 2n(n+1) - (n+1)^2 \\ &= 1 + 2n^2 + 2n - n^2 - 2n - 1 = n^2 \end{aligned}$$

This demonstrates that the program returns the squares sequence based on the sum of the result from the previous iteration with the consecutive odd integers, being valid for every $n \in \mathbb{N}_0$.

[4]Note that the sum of the first n odd numbers is n^2, that is, $\sum_{i=1}^{n} 2i - 1 = n^2$.

Modified factorial with varying steps. As mentioned earlier the reported results show that finding adequate solutions that memorise values from 3 or more iterations backwards is not an easy task for the algorithm, demonstrating some scalability issues in the representation. This is due to the increasing complexity of the programs when the step goes beyond 2, which is illustrated by the examples of minimal evolved solutions in Figure 6.17. One can see that when step 3 is reached there is practically the double of the minimum number of nodes and connections than in the previous step, close to 50% increase in the average number of functions, contrasting with the negligible difference from step 1 to 2 (see Table 6.6).

The generalisation values show that practically every solution found is able to correctly generate the first 100 elements of the sequence, with the exception of $s = 1$ (Table 6.6). As mentioned previously, the low rate of general programs observed with the smallest step may be an effect of early convergence, as the mean number of evaluations required to find a solution for the first ten elements is very small.

The generality of the solutions presented in Figure 6.17 can be manually verified, similarly to the previous problems. The simplified program for $s = 1$ is presented in Figure 6.18. Let the top node be f_n and the bottom subtraction node be g_n. The values for the first iteration can be directly calculated: $g_0 = 0$, and $f_0 = 1$. The logical expression of the nodes can then be written as presented in Equations 6.8 and 6.9.

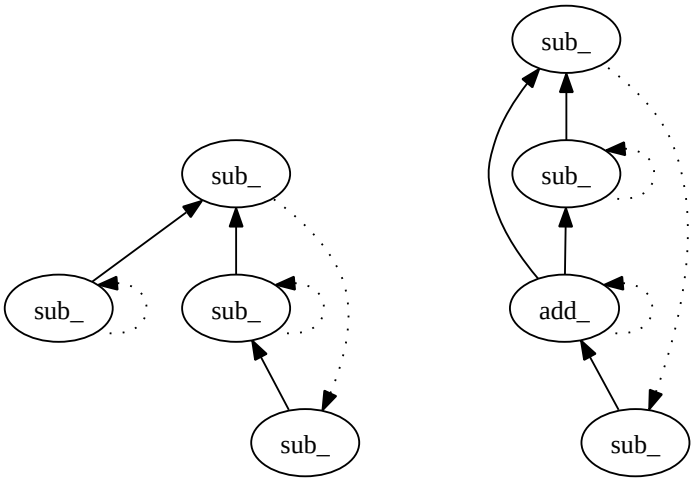
$$g_n = g_{n-1} - f_{n-1} \tag{6.8}$$

$$f_n = 1 - g_n \tag{6.9}$$

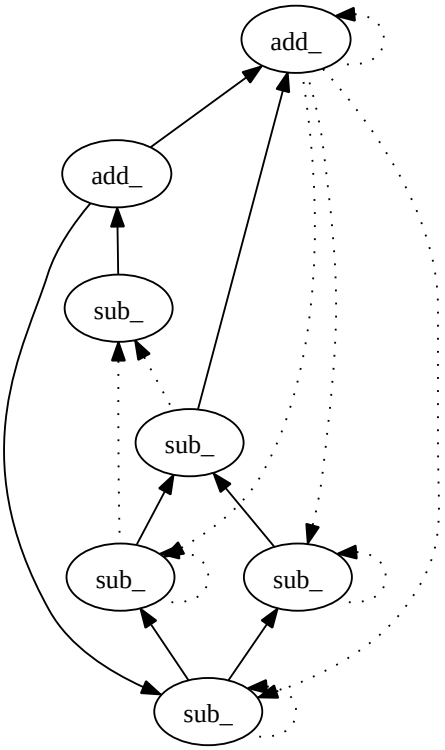
By telescoping g_n, it follows that:

$$\begin{aligned} g_n &= g_{n-1} - f_{n-1} \\ &= g_{n-2} - f_{n-2} - f_{n-1} \\ &= g_{n-k} - \sum_{i=1}^{k} f_{k-i} \end{aligned}$$

Then, for $k = n$, one has:

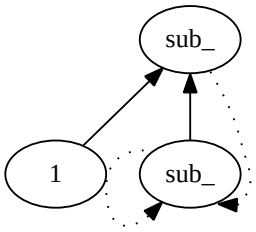


(a) step = 1. (b) step = 2.



(c) step = 3.

Figure 6.17: Examples of the smallest programs evolved that generates the fibonacci sequence for the step varying from 1 to 3. The nodes represent arithmetic operators. The output of the circuit is taken from the top. Each iteration the nodes are updated from bottom to top and the result is taken as an element of the sequence. Functions take their inputs from the nodes where their in-edges come from. The dotted edges represent feedback connections.



(a) step = 1.

Figure 6.18: The program from Fig. 6.17a, after simplification by removal of two functions with a single input.

$$g_n = g_0 - \sum_{i=1}^{n} f_{n-i} = -\sum_{i=1}^{n} f_{n-i}$$

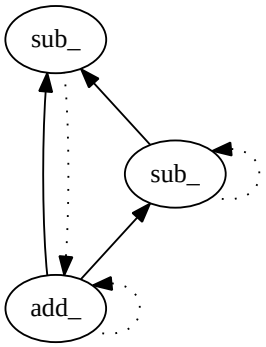
Replacing g_n in Equation 6.9, it holds:

$$\begin{aligned} f_n &= 1 + \sum_{i=1}^{n} f_{n-i} \\ &= 1 + \sum_{i=0}^{n-1} f_i \qquad (6.10) \\ f_{n-1} &= 1 + \sum_{i=0}^{n-2} f_i \qquad (6.11) \end{aligned}$$

If one inspects the difference between Eq. 6.11 and Eq. 6.10, we see that the program exhibits the correct behavior:

$$\begin{aligned} f_n - f_{n-1} &= \sum_{i=0}^{n-1} f_i - \sum_{i=0}^{n-2} f_i \\ f_n - f_{n-1} &= f_{n-1} \\ f_n &= 2f_{n-1} \end{aligned}$$

For the second instance the simplified program is presented in Figure 6.19. Let the top subtraction node be f_n, the bottom subtraction node be g_n, and the addition node be h_n. The logical expression of the nodes can then be written as presented in Equations 6.12-6.14.



(a) step = 2.

Figure 6.19: The program from Fig. 6.17b, simplified by the removal of a single input node.

$$h_n = f_{n-1} + h_{n-1} \tag{6.12}$$

$$g_n = h_n - g_{n-1} \tag{6.13}$$

$$\begin{aligned} f_n &= h_n - g_n \\ &= g_{n-1} \end{aligned} \tag{6.14}$$

Then, by replacing h_n in Equation 6.13, one has:

$$g_n = f_{n-1} + h_{n-1} - g_{n-1}$$

Since $h_{n-1} - g_{n-1} = f_{n-1}$ (Eq. 6.14), it follows that:

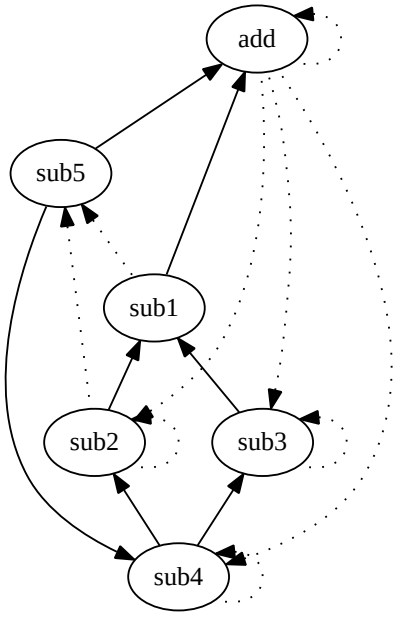
$$\begin{aligned} g_n &= 2f_{n-1} \\ f_n &= g_{n-1} \\ &= 2f_{n-2} \end{aligned}$$

For the last case, when the step is 3, a simplified version of the program is presented in Figure 6.20, and the simulation of the circuit depicted in Table 6.7 shows how the target sequence is produced through the first iterations.

The demonstration is not as straightforward as with the previous instances. As mentioned before, the recurrence relation that defines the modified factorial can be solved by telescoping, and in this case one ends up with:

Table 6.7: Simulation of the program graph execution for the modified factorial with step $s = 3$.

Iteration	Init	0	1	2	3	4	5	6	7	8	9	10	11	12
Node	Value													
add (output)	1	1	1	1	2	2	2	4	4	4	8	8	8	16
sub1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
sub2	1	0	0	1	0	0	2	0	0	4	0	0	8	0
sub3	1	0	0	1	0	0	2	0	0	4	0	0	8	0
sub4	1	0	1	0	0	2	0	0	4	0	0	8	0	0
sub5	1	0	0	0	1	0	0	2	0	0	4	0	0	8



(a) step = 3.

Figure 6.20: The program from Fig. 6.17c, simplified through the removal of a node with a single input.

$$F(n) = 2^{\lfloor \frac{n}{3} \rfloor}$$

From the data in Table 6.7 it is clear a pattern, and one can use a generating function to characterise the sequence.

$$G(z) = 1 + z + z^2 + 2z^3 + 2z^4 + 2z^5 + 4z^6 + 4z^7 + 4z^8 + 8z^9 + \ldots$$

Then, one can find the closed form for this generating function, as follows:

$$G(z) = \frac{1 + z + z^2}{1 - 2z^3}$$

Now, what is needed is to find the coefficient of order n, a_n, because $F(n) = a_n$. One way to do this is to expand $G(z)$ as a Taylor series, but that implies calculating the derivatives, which is a task not that simple. Nevertheless, one may decompose $G(z)$ using the technique of partial functions:

$$G(z) = \frac{1 + z + z^2}{1 - 2z^3} = \frac{1}{1 - 2z^3} + \frac{z}{1 - 2z^3} + \frac{z^2}{1 - 2z^3} = A(z) + zA(z) + z^2A(z)$$

So, each element of the original sequence may be obtained as the sum of three components. A closer inspection shows that the second operand is the right shift of the first, and the third operand is the right shift twice of the first operand. Let's see what the expansion of $A(z)$ looks like:

$$A(z) = 1 + 2z^3 + 4z^6 + 8z^9 + 16z^{12} + \ldots$$

and the sequence is:

$$1, 0, 0, 2, 0, 0, 4, 0, 0, 8, 0, 0, 16, \ldots$$

It only outputs non-zero values when the exponent is a multiple of three. Using that information one can easily come to the conclusion that when $n = 3k, \quad k = 0, 1, 2, 3, \ldots$ the coefficient of order n is $2^{\frac{n}{3}}$. Now, all one has to do is to repeat the process with the other two components, which is easy because they result of a shift right operation of $A(z)$.

$$A(z) : 1, 0, 0, 2, 0, 0, 4, 0, 0, 8, 0, 0, 16, \ldots$$
$$zA(z) : 0, 1, 0, 0, 2, 0, 0, 4, 0, 0, 8, 0, 0, \ldots$$
$$z^2A(z) : 0, 0, 1, 0, 0, 2, 0, 0, 4, 0, 0, 8, 0, \ldots$$

It becomes clear that, for a particular n, only one component is non zero. Now suppose we are considering a value, n, that is a multiple of three. In that situation only the $A(z)$ component will be non zero and we have $a_n = 2^{\frac{n}{3}}$. If we choose a number whose antecessor, of order n, is a multiple of three, only the $zA(z)$ component will be non zero, and again one has that $a_{n+1} = 2^{\frac{n+1}{3}}$. Using the same reasoning for a number that is equal to $n + 2$ we come also to the conclusion that $a_{n+2} = 2^{\frac{n+2}{3}}$. Summing up the three possible situations, we can say that $\forall n = 0, 1, 2, 3, \ldots : a_n = 2^{\lfloor \frac{n}{3} \rfloor}$. This shows that our program computes the target $F(n)$.

7 ReNCoDe with Multiple Outputs

The ReNCoDe model described in Chapter 3 is not ready to build programs that are able to deal with multiple parallel outputs. In order to be able to address this class of problems, the model was adapted by integrating an extended model of the ARN, with input and output proteins as proposed by [Nicolau et al., 2010]. The remaining of this chapter describes the modifications to the architecture, as well as the problems used as benchmark, and presents the results achieved.

7.1 Extending ReNCoDe

The merge of ReNCoDe (see Section 3.2) with the extended ARN model presented in [Nicolau et al., 2010] involves three aspects: the definition of the inputs and the outputs, the algorithm for extracting the program from the network, and the mapping of proteins to functions. The proof of concept for this model was presented in [Lopes and Costa, 2013b].

Input and Output

The original proposal of the ARN models a regulatory network as a closed self-contained system. In order to extend the model with input/output capabilities two directions were taken by [Nicolau et al., 2010]. First, extra proteins were introduced to represent the inputs (receptors), which are not coded in the genome. Second, the proteins coded by the genome are distinguished between transcription factors (TFs) and products, based on

using two promoters *XYZ*00000000 and *XYZ*11111111, respectively. The receptors have distinct 32-bit signatures as presented in the extended ARN (see Section 3.1). If there are more than four inputs, for instance the following signatures may be used alongside the ones presented before:

00000000111111110000000011111111
11111111000000001111111100000000
00001111000011110000111100001111
11110000111100001111000011110000

The receptors participate in the regulation mechanism, thus they can bind to the regions upstream of the genes. As these are external to the network and are not coded in the genome, they are not regulated. The products are regulated by all proteins but do not regulate. This means that, in the resulting network, there are no connections towards the inputs, nor from the output(s) towards other proteins (see Fig. 3.5).

The number of products encoded in a genome is variable, as it depends on the occurrence of the specific promoter. In this variant of ReNCoDe, only the necessary amount of products is used, and the remaining are discarded. The next section illustrates how to build a program graph from this ARN model, and clarifies how the products are used as outputs.

Extracting the program graph

First the ARN of the individual is built, composed of multiple links (inhibition and excitation) between different nodes (genes). In order to extract a graph from this network it must first be reduced. This is achieved with Algorithm 1, described in Section 3.2. An ARN with inputs and outputs, as well as the corresponding reduced graph is presented in Figure 7.1. In this example one can see that after reducing the network node 6 is not input to node 2 anymore, meaning that the inhibition from node 6 to node 2 is stronger than excitation and thus was discarded.

The graph is constructed in a top-down fashion, starting from the output node(s). The process is illustrated by the pseudo-code in Algorithm 5. When only one output is necessary each product is tested as the graph output (a graph is built and tested for each

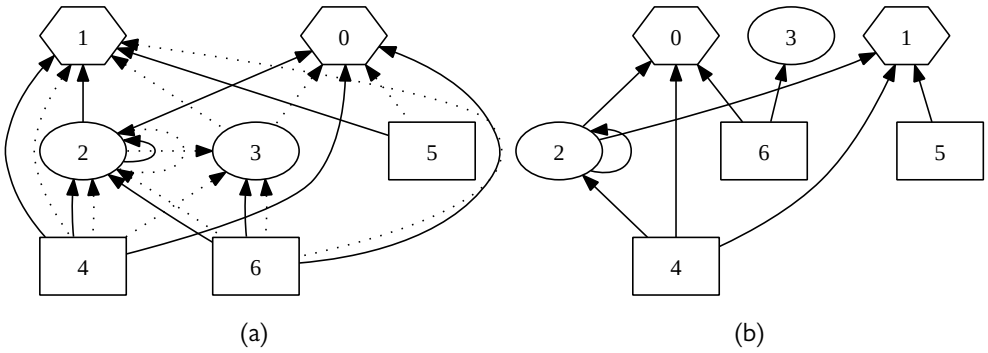


Figure 7.1: a) Extended ARN example. The dotted and full edges represent, respectively, inhibition and excitation relationships between genes. The numbers are just identifiers. The hexagonal nodes correspond the output proteins, the rectangular represent the extra input proteins, and the oval nodes correspond to the transcription factors. b) The graph obtained from the same network after application of Algorithm 1.

product). If one wants N outputs from the graph, then the first N products are used as outputs and the graph is built from these in the same fashion (there are no connections amongst the outputs since they do not regulate). The program graph is initialised by mapping the products into the output functions. Then a queue is initialised with the nodes which are input to the output nodes - corresponding to the TFs in the extended ARN model (lines 1-2).

Then , the following process is repeated until the queue is empty and there are no TFs left to map (lines 3-13). First, the queue is sorted by the edge strength[1]. The top node on the queue is mapped into a function and added to the final program. Any TFs that are input to this node are also added to the queue. Then, the node is removed from the queue, and the next on the queue can be processed. Finally, when the queue is empty, the extra input proteins are added to the program graph, mapped into the corresponding input value/variable (line 14).

Recursive connections are avoided during the mapping of the node into a program function (line 5) similarly to the original model. The mapping process is detailed in the next section.

Figure 7.2 shows an example of a program with two outputs, built from the reduced

[1] As mentioned in Section 3.2, the ARN network graph is reduced by transforming the inhibition and excitation edges into a single edge with the difference $(e-h)$. This is the value that is used to sort the input edges.

Algorithm 5 Program extraction for the extended ReNCoDe

```
1: initialise program with output functions
2: initialise queue with the inputs for the output functions
3: while queue is not empty do
4:    sort nodes by matching strength
5:    map top node in queue to function
6:    add function to program
7:    for all input in top node do
8:       if not in program and not a receptor then
9:          add input to queue with respective strength
10:      end if
11:   end for
12:   remove the top node from queue
13: end while
14: add receptors to program
```

graph presented in Figure 7.1b. Note that node 3 has disappeared since it was not input to another node. If there are not enough products in a network, 0 will be returned for the missing outputs.

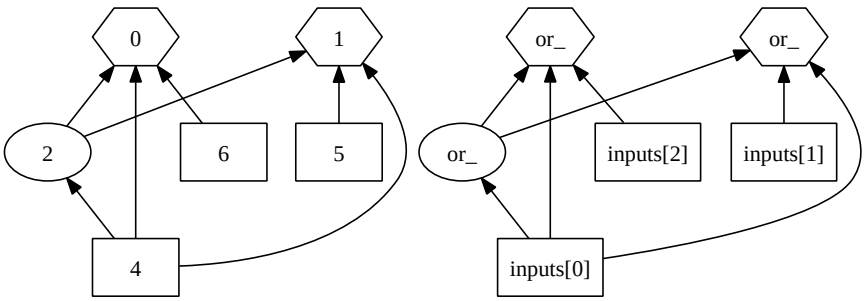


Figure 7.2: a) The executable graph extracted from the reduced network depicted in Fig. 7.1b. Functions take their inputs from the nodes where their in-edges come from, and the labels are identifiers. The hexagonal nodes correspond the output proteins, the rectangular represent the extra input proteins, and the oval nodes correspond to the transcription factors. b) The same graph after the mapping process.

Mapping Proteins to Functions

A mapping is needed to translate nodes (i.e., genes/proteins) to functions. The terminals are mapped sequentially using the input proteins (disregard of their signature). For instance, if a problem has only four inputs, they are mapped as follows (*proteinsignature* $\rightarrow$ *input*):

$$
\begin{aligned}
&00000000000000000000000000000000 \rightarrow inputs[0]\\
&00000000000000001111111111111111 \rightarrow inputs[1]\\
&11111111111111110000000000000000 \rightarrow inputs[2]\\
&11111111111111111111111111111111 \rightarrow inputs[3]
\end{aligned}
$$

If a problem has more than four inputs then it is necessary to provide more input proteins with distinct signatures, as shown before. In the support library this model has been implemented to handle from one to twelve inputs. For the remaining nodes we use the gene-protein correspondence, as described in the original ReNCoDe, obtaining the function by a majority vote process. Differently from the original model it is possible for some node to be included in the graph, while not having any input connection available. In this case the proteins' signature is mapped to the interval $[0, 1]$ and rounded to the closest integer, generating either a 0 or a 1. This way, the mapping is valid either as a decimal or a boolean value. The pseudo-code in Algorithm 6 illustrates this mapping algorithm.

7.2 Problems

[White et al., 2012] propose the full adder and the multiplier as the reference benchmarks for boolean logic. Multiple output programs are common in real-world applications, but are usually avoided by researchers in GP. Typically, as one increases the number of inputs and outputs the problem instances become exponentially harder.

The Binary Adder

In general terms, a binary adder takes two n-bit inputs, and a 1-bit carry-in totalling $2n+1$ inputs. Likewise it returns a n-bit sum, and a 1-bit carry-out, giving a total of $n+1$ outputs.

Algorithm 6 Mapping nodes to program functions for the variant with multiple outputs.

```
1: filter node-inputs already in program
2: if node is not a receptor then
3:    if node-inputs not empty then
4:       funindex ← apply majorityrule to protein-signature
5:       map node to corresponding function
6:    else {node-inputs is empty}
7:       value ← map protein-signature to float
8:       map node to round(value)
9:    end if
10: else {node is a receptor}
11:    map node into corresponding input variable/value
12: end if
```

The full adder is a logical unit that performs the sum of two 1-bit inputs (A and B), taking into account the carry-in bit (C_{in}) and outputs the sum (S) and the carry-out (C_{out}). The block diagram for the 1-bit full adder is presented in Figure 7.3, and the corresponding truth table is given in Table 7.1.

Table 7.1: The truth table for the full adder.

A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

This problem has not been extensively studied in GP. Traditionally specific instances are tackled [Koza and Rice, 1991, Miller et al., 1998, Sen, 1998, Walker and Miller, 2005b, Kuyucu et al., 2009b], but the most interesting approaches to this problem use artifi-

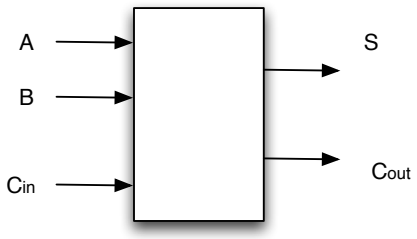


Figure 7.3: Block diagram for the 1-bit full adder.

cial development to achieve scalability, evolving circuits that grow providing n-bit adders [Gordon and Bentley, 2005, Harding et al., 2010].

The function set used throughout the literature vary, with some functions providing shortcuts to the successful evolution of circuits. In ReNCoDe the traditional function set $\{and, or, nand, nor\}$ is used, while the terminal set is composed of $\{A, B, C_{in}\}$. Typically, the fitness measure used is the number of wrong bits in the output of the solutions.

To the best of our knowledge, [Walker and Miller, 2004b] present the best approach, reaching a success rate of 100%, and a median number of evaluations below 10000 (for both CGP and Embedded-CGP). Moreover, they also address the 2-bit and 3-bit instances.

The Binary Multiplier

The binary multiplier takes two n-bit inputs, and returns their product as a $2n$-bit output. The simplest instance of this problem is the 2-bit multiplier. This circuit multiplies two 2-bit inputs (A and B), resulting in a 4-bit output (P). The block diagram is presented in Figure 7.4, and the corresponding truth table is given in Table 7.2.

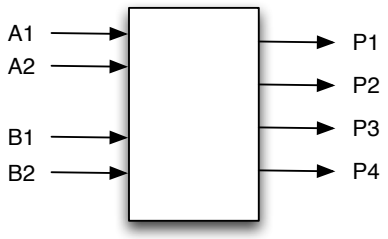


Figure 7.4: Block diagram for the 2-bit multiplier.

Table 7.2: The truth table for the 2-bit multiplier.

A_1	A_2	B_1	B_2	P_4	P_3	P_2	P_1
0	0	0	0	0	0	0	0
		0	1	0	0	0	0
		1	0	0	0	0	0
		1	1	0	0	0	0
0	1	0	0	0	0	0	0
		0	1	0	0	0	1
		1	0	0	0	1	0
		1	1	0	0	1	1
1	0	0	0	0	0	0	0
		0	1	0	0	1	0
		1	0	0	1	0	0
		1	1	0	1	1	0
1	1	0	0	0	0	0	0
		0	1	0	0	1	1
		1	0	0	1	1	0
		1	1	1	0	0	1

The literature shows that this is an interesting problem with challenging fitness landscapes [Vassilev et al., 1999]. Despite work on the design of multiplier circuits being scarce, there are some proposals, for instance [Kuyucu et al., 2009b, Miller, 1999, Vassilev and Miller, 2000, Walker and Miller, 2005a, Poli, 1998, Helmuth and Spector, 2013].

Similarly to the full adder approaches, varied function sets are used throughout the proposals. In this case the same function set as in the full-adder was used $\{and, or, nand, nor\}$, and the terminal set is composed of $\{A_1, A_2, B_1, B_2\}$. The typical fitness function is the number of incorrect bits in the outputs, similarly to the previous problem.

The best approaches to this boolean regression problem that are more close to traditional GP are reported by [Walker and Miller, 2004b], again achieving a success rate of 100% on the simplest instance, with the median number of evaluations below 10000. Moreover, the 3-bit instance was also solved although with much higher computational effort. [Helmuth and Spector, 2013] also report a 99% success rate for the 2-bit multiplier, but were not able to scale up.

7.3 Experiments and Results

The parameterisation of the base ES is the same as described before (please refer to Table 5.1 for the details). However, the ARN configuration used in the previous experiments was not adequate for these problems. Consequently, some trial and error was necessary to find a better configuration for the parameters. In both cases the first instances of the problems were tackled: the 1-bit adder (a.k.a. full adder), and the 2-bit multiplier (since the 1-bit multiplier is simply the *and* function).

Using randomly initialised genomes with 2^{11} bits of length, 90% of the runs were able to find a solution for the full-adder problem. Other configurations with smaller genome sizes were also able to find solutions, but with lower success rates. In the case of the multiplier it was not possible to find solutions using the typical fitness measure and function set. Nevertheless, some solutions were evolved for this problem using the same function set of the adder and randomly initialised genomes with 2^7 bits of length, with a success rate of 3%, using as the fitness function the number of incorrect outputs (in contrast to the number of incorrect bits in the outputs).

These results are summarised in Table 7.3, showing that this variant of ReNCoDe is able to generate programs for multiple output binary logic. Despite being able to evolve correct programs, the success rates are low and too many evaluations are necessary before an optimum is evolved. In Figure 7.5 an example of the evolved solutions is presented for each problem.

Table 7.3: Summary of the results for the multiple output problems.

Problem	Adder	Multiplier
% of Successful Runs	90	3
Min. Numb. of Evaluations	31300	352600
Avg. Numb. of Evaluations (Std. Dev.)	2160448 (2097275)	1139067 (714228)

The binary adder depicted can be described by the following equations:

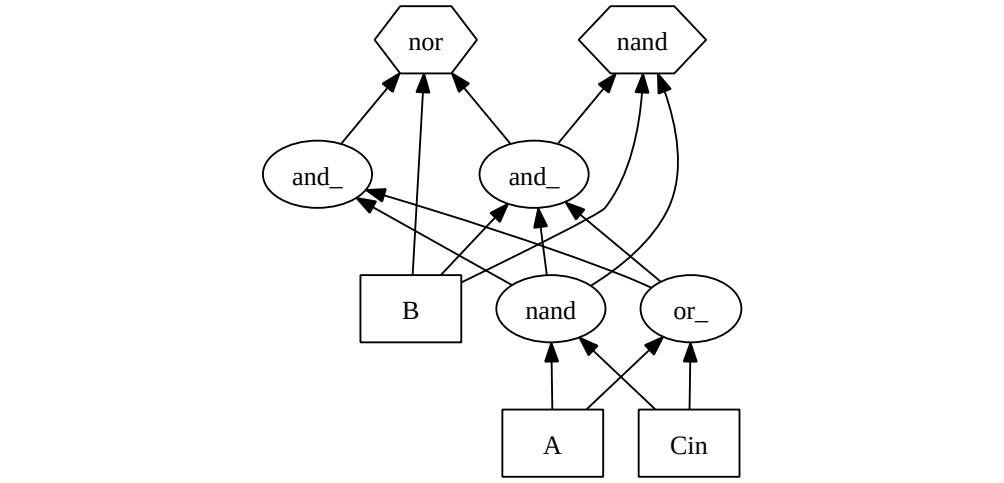
$$S = (((A \vee C_{in}) \wedge (A \barwedge C_{in})) \veebar B) \veebar ((B \wedge (A \barwedge C_{in})) \wedge (A \vee C_{in}))$$
$$C_{out} = (((B \wedge (A \barwedge C_{in})) \wedge (A \vee C_{in})) \barwedge B) \barwedge (A \barwedge C_{in})$$

The equations that follow in turn translate and simplify the multiplier displayed in the

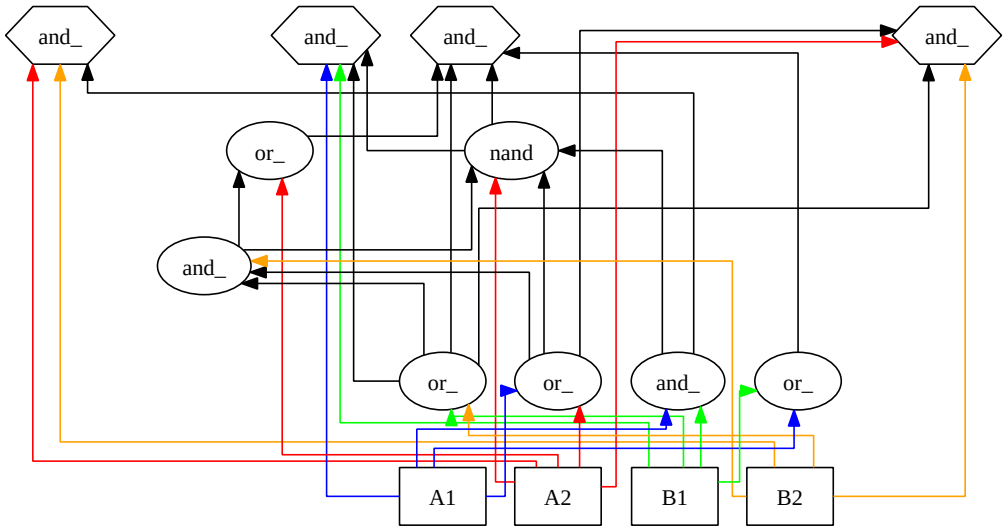
same figure:

$$P_4 = A_1 \wedge A_2 \wedge B_1 \wedge B_2$$
$$P_3 = A_1 \wedge \neg(A_2 \wedge B_2) \wedge B_1$$
$$P_2 = (\neg A_1 \vee \neg A_2 \vee \neg B_1 \vee \neg B_2) \vee (A_1 \vee A_2) \wedge (A_1 \vee B_1) \wedge (A_2 \vee B_2) \wedge (B_1 \vee B_2)$$
$$P_1 = A_2 \wedge B_2$$

Multiple output binary logic problems are considered to be a difficult class of problems. Despite being easy to extend the representation to deal with multiple outputs, and the fact that correct solutions were evolved to these problems, the performance achieved so far has room for improvement. Typically the difficulty increases exponentially with the increase of the size of the inputs, and with ReNCoDe this was not an exception.



(a) Adder



(b) Multiplier

Figure 7.5: Examples of the simplest programs evolved that solve each of the multiple output problems presented. The nodes represent binary operators. The output of the circuit is taken from the hexagon shaped nodes. Functions take their inputs from the nodes where their in-edges come from. a) Adder: the outputs correspond to the sum (S) and C_{out}, respectively from left to right. b) Multiplier: the outputs correspond to P_4, P_3, P_2, and P_1, respectively from left to right; each input has the connections coloured differently for easy tracking.

8

Conclusion and Future Work

While the pioneers of Artificial Intelligence (AI) envisioned fast progress towards real machine intelligence, most of their prophecies still belong to the realm of science fiction. Nonetheless, progress has been made, and AI is nowadays a huge interdisciplinary field of research, with applications in the most varied domains.

Much of the inspiration for AI systems comes from natural organisms (like bacteria, viruses, the human brain, colonies of insects and more), shaped by evolution over millions of years. Some researchers used evolution itself as an analogy to breed solutions for optimisation problems and even problem solvers, in the computational world. Evolution Strategies, Evolutionary Programming, and Genetic Algorithms were the first Evolutionary Computation (EC) methodologies, followed by Genetic Programming which has special relevance in this thesis.

These methodologies rely on the concept of natural selection as presented by C. Darwin more than one century ago, and the consequent Modern Synthesis of Evolution. However, it is well known that there is much more to evolution than just natural selection. The knowledge on the biological mechanisms that shape life has greatly evolved since Francis and Crick proposed the physical medium for inheritance. However, most of it has been disregarded amongst the EC practitioners.

The main issues pointed out to the traditional approaches are the over-simplification of the genotype-phenotype relationship (one-to-one mapping), the absence of developmental processes, or the absence of interaction between the individuals and the environment. Behind these issues there is a common mechanism, the Genetic Regulatory Network (GRN), formed by the interactions between the genes and the proteins. The

activity of each gene is influenced not only by the other genes and their products, but also by signalling molecules coming from neighbouring cells and the environment. It is the differences in gene expression, and the interaction with the environment, that allow the myriad of different species we have on the blue planet.

The working hypothesis in this thesis is that the inclusion of GRN models in Evolutionary Algorithms (EAs) may not only improve its efficacy and efficiency in benchmark results, but also broaden the classes of problems that can be addressed.

In this work a new EA was developed, which uses an Artificial Regulatory Network (ARN) as the genotypic representation, coupled with an algorithm to reduce the complexity of the network graph and map it into an executable program: the Regulatory Network Computational Device (ReNCoDe). A variant of the algorithm with embedded memory was also developed, extending the capabilities of the approach to a different class of problems. Moreover, this work also contributes with biologically inspired asexual variation operators, designed to deal with linear genotypes, and also specific for the ARN model. These were inspired by the biological concepts of *transposons* and *junk-DNA*.

ReNCoDe was validated with a set of traditional benchmarks from symbolic regression and control, to learning. The ARN model was tuned using this set of problems, and the results show that ReNCoDe is competitive with other methodologies. The success rates with these problems are around 100%, and it even outperforms other approaches in the number of evaluations necessary to find a solution, as well as in the generalisation capabilities of the evolved programs. Moreover, the bloat present in the solutions is practically insignificant, resulting in simple program graphs that can be later analysed and manipulated by the researcher.

The efficiency of the variation operators (both sexual and asexual) was also evaluated using the same benchmarks, and compared with the baseline from the previous experiments. Different configurations and rates of application were tested, but none of them performed significantly better than the baseline.

The variant with feedback connections was tested with a set of recursive benchmarks, using the best parameterisation from the previous results. ReNCoDe was able to solve all the problems, achieving 100% of success rate, and in most of the problems every so-

lution found was able to pass the generalisation test. The results are competitive with the reports in the literature, even outperforming them in some cases. Moreover, the simplicity of the evolved solutions allows the analysis and formalisation of the corresponding programs, as demonstrated with the examples for each of the recursive problems. There seems to be however an Achilles' heel in this representation: the experiments with the modified factorial have shown a scalability issue. When it is necessary to access results from three (or more) iterations earlier, evolution is difficult and often unsuccessful.

Even though ReNCoDe has been tested over several classes of problems, there are other classes of problems left to address. One example are programs with multiple outputs. An initial study of this class of problems was also presented. The full adder and the 2-bit multiplier were investigated with a variant of the original ReNCoDe algorithm. This variant uses an extended model of the ARN, that discriminates the proteins encoded in the genome between products and transcription factors. The results are promising, with both problems solved, but with room for improvement in performance. The multiple-output domain is not limited to binary problems, thus future work should research other domains, like symbolic regression and classification. Another topic of interest nowadays are the dynamic environments, where the properties of the system change through time during the evolutionary process, which is the case of most of the real-world problems. Clearly the interaction with the environment must have special relevance in this domain, and this is a point that is still missing in most of the GRN models. Thus, this is another focal point for future work: how to include the interaction with the environment without loosing the generality of the approach.

The ARN model was the starting point for the architecture described in this work, a sensible choice since it has been published for about a decade and it has been studied from several points of view by other researchers. Despite its completeness and accordance to the natural counterpart, some aspects are missing that only in the last decade were revealed by the Biological Sciences. A clear example of that is epigenetics, a mechanism that is still not well-known, but for which there are already some models that were able to mimic the natural behaviour. With that in mind, future work should try to include epigenetics into the ARN, and test different models that already have some epigenetic mechanism implemented.

The founders of the AI research field were rather optimistic concerning the developments that would be achieved in the first decades of research towards machine intelligence. Despite all the science-fiction around the field, the rather slow start proved them wrong, as researchers realised that just like Turing had adverted before "we can only see a short distance ahead, but we can see plenty there that needs to be done".

Bibliography

[Banzhaf, 2003] Banzhaf, W. (2003). On the Dynamics of an Artificial Regulatory Network. In Banzhaf, W., Ziegler, J., Christaller, T., Dittrich, P., and Kim, J., editors, *Lecture Notes in Computer Science*, pp 217–227. Springer Berlin / Heidelberg.

[Banzhaf et al., 2006] Banzhaf, W., Beslon, G., Christensen, S., Foster, J. A., Képès, F., Lefort, V., Miller, J. F., Radman, M., and Ramsden, J. J. (2006). Guidelines: From artificial evolution to computational evolution: a research agenda. *Nature Reviews Genetics*, 7(9):729–735.

[Bentley, 2004] Bentley, P. J. (2004). Fractal Proteins. *Genetic Programming and Evolvable Machines*, 5(1):71–101.

[Brameier and Banzhaf, 2007] Brameier, M. F. and Banzhaf, W. (2007). *Linear Genetic Programming*. Springer.

[Bray, 2009] Bray, D. (2009). *Wetware*. A Computer in Every Living Cell. Yale University Press.

[Byrne et al., 2009] Byrne, J., O'Neill, M., and Brabazon, A. (2009). Structural and nodal mutation in grammatical evolution. In Raidl, G., Rothlauf, F., Squillero, G., Drechsler, R., Stuetzle, T., Birattari, M., Congdon, C. B., Middendorf, M., Blum, C., Cotta, C., Bosman, P., Grahl, J., Knowles, J., Corne, D., Beyer, H.-G., Stanley, K., Miller, J. F., van Hemert, J., Lenaerts, T., Ebner, M., Bacardit, J., O'Neill, M., Di Penta, M., Doerr, B.,

Jansen, T., Poli, R., and Alba, E., editors, *GECCO '09: Proceedings of the 11th annual Conference on Genetic and Evolutionary Computation*, pp 1881–1882, Montreal. ACM.

[Carroll, 2006] Carroll, S. B. (2006). *Endless Forms Most Beautiful: The New Science of Evo Devo*. W. W. Norton & Company.

[Carroll, 2007] Carroll, S. B. (2007). *The Making of the Fittest: DNA and the Ultimate Forensic Record of Evolution*. W. W. Norton & Company.

[Casula et al., 2009] Casula, G. A., Mazzarella, G., and Sirena, N. (2009). Genetic Programming design of wire antennas. In *IEEE Antennas and Propagation Society International Symposium, APSURSI '09*, pp 1–4.

[Choi et al., 2007] Choi, H. S., Jung, S. H., Kim, J., Cho, K. H., Kim, J. R., and Choo, S. M. (2007). Reverse engineering of gene regulatory networks. *IET Systems Biology*, 1(3):149–163.

[Christensen and Oppacher, 2007] Christensen, S. and Oppacher, F. (2007). Solving the artificial ant on the Santa Fe trail problem in 20,696 fitness evaluations. *GECCO '07: Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation*, pp 1574–1579.

[Crombach and Hogeweg, 2008] Crombach, A. and Hogeweg, P. (2008). Evolution of Evolvability in Gene Regulatory Networks. *PLoS Computational Biology*, 4(7).

[Cussat-Blanc et al., 2011] Cussat-Blanc, S., Bredeche, N., Luga, H., and Duthen, Y. (2011). Artificial gene regulatory networks and spatial computation: A case study. *Advances in Artificial Life: European Conference on Artificial Life 2011*.

[Davidson, 2010] Davidson, E. H. (2010). *The Regulatory Genome*. Gene Regulatory Networks In Development And Evolution. Academic Press.

[Dawkins, 2006] Dawkins, R. (2006). *The Selfish Gene: 30th Anniversary edition*. Oxford University Press.

[Dawkins, 2008] Dawkins, R. (2008). *The Oxford Book of Modern Science Writing*. Oxford University Press.

[de Jong, 2002] de Jong, H. (2002). Modeling and Simulation of Genetic Regulatory Systems: A Literature Review. *Journal of Computational Biology*, 9(1):67–103.

[Eiben and Smith, 2003] Eiben, A. E. and Smith, J. E. (2003). *Introduction to Evolutionary Computing*. Springer-Verlag.

[Esparcia-Alcazar and Sharman, 1997] Esparcia-Alcazar, A. I. and Sharman, K. (1997). Evolving Recurrent Neural Network Architectures by Genetic Programming. In Koza, J. R., Deb, K., Dorigo, M., Fogel, D. B., Garzon, M., Iba, H., and Riolo, R. L., editors, *Genetic Programming 1997: Proceedings of the second annual Conference on Genetic Programming*, pp 89–94, Stanford University, CA, USA. Morgan Kaufmann.

[Ferreira, 2002] Ferreira, C. (2002). Genetic representation and genetic neutrality in gene expression programming. *Advances in Complex Systems*, 5(04):389–408.

[Field, 2003] Field, A. P. (2003). *How to Design and Report Experiments*. Sage Publications Ltd.

[Filkov, 2005] Filkov, V. (2005). Identifying gene regulatory networks from gene expression data. *Handbook of Computational Molecular Biology*, pp 27.1–27.29.

[Fogel, 2006] Fogel, D. B. (2006). *Evolutionary Computation Toward a New Philosophy of Machine Intelligence*. Wiley-IEEE Press, Hoboken, 3rd edition.

[Fogel and Burgin, 1969] Fogel, L. J. and Burgin, G. H. (1969). Competitive goal-seeking through evolutionary programming. Technical report, Defense Technical Information Center.

[Freeman, 1998] Freeman, J. J. (1998). A linear representation for GP using context free grammars. *Genetic Programming 1998: Proceedings of the third annual Conference on Genetic Programming*, pp 72–77, Morgan Kaufmann.

[Geard and Willadsen, 2009] Geard, N. N. and Willadsen, K. K. (2009). Dynamical approaches to modeling developmental gene regulatory networks. *Birth Defects Research Part A: Clinical and Molecular Teratology*, 87(2):131–142.

[Gordon and Bentley, 2005] Gordon, T. G. and Bentley, P. J. (2005). Development brings scalability to hardware evolution. *Proceedings of NASA/DoD Conference on Evolvable Hardware*, pp 272–279.

[Harding and Banzhaf, 2008] Harding, S. and Banzhaf, W. (2008). Artificial Development. In Würtz, R. P., editor, *Organic Computing*, pp 201–220. Springer-Verlag.

[Harding et al., 2009a] Harding, S., Miller, J. F., and Banzhaf, W. (2009a). Self Modifying Cartesian Genetic Programming: Fibonacci, Squares, Regression and Summing. In *EuroGP '09: Proceedings of the 12th European Conference on Genetic Programming*. Springer-Verlag.

[Harding et al., 2009b] Harding, S., Miller, J. F., and Banzhaf, W. (2009b). Self modifying Cartesian Genetic Programming: Parity. *Congress on Evolutionary Computation, CEC 2009*, pp 285–292.

[Harding et al., 2010] Harding, S., Miller, J. F., and Banzhaf, W. (2010). Developments in Cartesian Genetic Programming: self-modifying CGP. *Genetic Programming and Evolvable Machines*, 11(3-4):397–439.

[Harding et al., 2007] Harding, S. L., Miller, J. F., and Banzhaf, W. (2007). Self-modifying cartesian genetic programming. *GECCO '07: Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation*, pp 1021–1028.

[Hartl, 2014] Hartl, D. L. (2014). *Essential Genetics*. A Genomics Perspective. Jones & Bartlett Publishers.

[Hecker et al., 2009] Hecker, M., Lambeck, S., Töpfer, S., van Someren, E. P., and Guthke, R. (2009). Gene regulatory network inference: Data integration in dynamic models - A review. *Biosystems*, 96(1):86–103.

[Helmuth and Spector, 2013] Helmuth, T. and Spector, L. (2013). Evolving a digital multiplier with the pushgp genetic programming system. In *GECCO '13: Proceedings of the 15th annual Conference on Genetic and Evolutionary Computation*, pp 1627–1634. ACM.

[Hohil et al., 1999] Hohil, M. E., Liu, D. R., and Smith, S. H. (1999). Solving the N-bit parity problem using neural networks. *Neural Networks*, 12(9):1321–1323.

[Holland, 1973] Holland, J. H. (1973). Genetic Algorithms and the Optimal Allocation of Trials. *SIAM Journal on Computing*, 2(2):88–105.

[Igel and Toussaint, 2003] Igel, C. and Toussaint, M. (2003). Neutrality and self-adaptation. *Natural Computing*, 2(2):117–132. Springer.

[Jablonka and Lamb, 2005] Jablonka, E. and Lamb, M. J. (2005). *Evolution In Four Dimensions*. Genetic, Epigenetic, Behavioral, And Symbolic Variation In The History Of Life. MIT Press.

[Keijzer, 2003] Keijzer, M. (2003). Improving symbolic regression with interval arithmetic and linear scaling. *EuroGP 2003: Proceedings of the 6th European Conference on Genetic Programming*, pp 70–82. Springer.

[Koza, 1992] Koza, J. R. (1992). *Genetic Programming*. On the Programming of Computers by Means of Natural Selection. The MIT Press.

[Koza, 1994] Koza, J. R. (1994). *Genetic programming II*. Automatic Discovery of Reusable Programs. The MIT Press.

[Koza, 2010] Koza, J. R. (2010). Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines*, 11(3-4):251–284. Springer.

[Koza et al., 1997a] Koza, J. R., Andre, D., Bennett III, F. H., and Keane, M. A. (1997a). Design of a high-gain operational amplifier and other circuits by means of genetic programming. In Angeline, P. J., Reynolds, R. G., McDonnell, J. R., and Eberhart, R., editors, *Evolutionary Programming VI. 6th International Conference, EP97*, volume 1213 of *Lecture Notes in Computer Science*, pp 125–136, Indianapolis, Indiana, USA. Springer-Verlag.

[Koza et al., 1997b] Koza, J. R., Bennett III, F. H., Andre, D., and Keane, M. A. (1997b). Evolution Using Genetic Programming of a Low-Distortion 96 Decibel Operational Amplifier. In Bryant, B., Carroll, J., Oppenheim, D., Hightower, J., and George, K. M., editors, *Proceedings of the 1997 ACM Symposium on Applied Computing*, pp 207–216, Hyatt Sainte Claire Hotel, San Jose, California, USA.

[Koza et al., 1998] Koza, J. R., Bennett III, F. H., Andre, D., and Keane, M. A. (1998). Evolutionary Design of Analog Electrical Circuits using Genetic Programming. In *Proceedings of Adaptive Computing in Design and Manufacture Conference*, Plymouth, England.

[Koza and Rice, 1991] Koza, J. R. and Rice, J. P. (1991). Genetic generation of both the weights and architecture for a neural network. *International Joint Conference on Neural Networks 1991, IJCNN'91*, 2:397–404. IEEE.

[Krohn et al., 2009] Krohn, J., Bentley, P., and Shayani, H. (2009). The challenge of irrationality: fractal protein recipes for PI. *GECCO'09: Proceedings of the 11th annual Conference on Genetic and Evolutionary Computation*, pp 715–722. ACM.

[Krohn and Gorse, 2010] Krohn, J. and Gorse, D. (2010). Fractal gene regulatory networks for control of nonlinear systems. In *PPSN'10: Proceedings of the 11th Conference on Parallel Problem Solving from Nature*, pp 209–218. Springer.

[Krohn and Gorse, 2012] Krohn, J. and Gorse, D. (2012). Extracting Key Gene Regulatory Dynamics for the Direct Control of Mechanical Systems. *PPSN'12: Proceedings of the 12th Conference on Parallel Problem Solving from Nature*, pp 468–477. Springer.

[Kumar and Bentley, 2003] Kumar, S. and Bentley, P. J. (2003). *On Growth, Form and Computers*. Academic Press.

[Kuo et al., 2006] Kuo, P. D., Banzhaf, W., and Leier, A. (2006). Network topology and the evolution of dynamics in an artificial genetic regulatory network model created by whole genome duplication and divergence. *Biosystems*, 85(3):177–200. Elsevier.

[Kuo et al., 2004] Kuo, P. D., Leier, A., and Banzhaf, W. (2004). Evolving dynamics in an artificial regulatory network model. *PPSN'04: Proceedings of the 8th Conference on Parallel Problem Solving from Nature*, 3242:571–580. Springer.

[Kuyucu et al., 2009a] Kuyucu, T., Trefzer, M. A., Miller, J. F., and Tyrrell, A. M. (2009a). A scalable solution to n-bit parity via artificial development. Research in Microelectronics and Electronics, 2009. PRIME 2009. Ph.D., pp 144–147. IEEE.

[Kuyucu et al., 2009b] Kuyucu, T., Trefzer, M. A., Miller, J. F., and Tyrrell, A. M. (2009b). Task decomposition and evolvability in intrinsic evolvable hardware. In *IEEE Congress on Evolutionary Computation, CEC'09*, pp 2281–2287. IEEE.

[Lane, 2010] Lane, N. (2010). *Life Ascending: The Ten Great Inventions of Evolution*. W. W. Norton & Company.

[Langdon and Banzhaf, 2000] Langdon, W. B. and Banzhaf, W. (2000). Genetic Programming Bloat without Semantics. In *PPSN'00: Proceedings of the 6th Conference on Parallel Problem Solving from Nature*. Lecture Notes in Computer Science (Vol. 1917), pp 201–210. Springer-Verlag.

[Lohn et al., 2004] Lohn, J., Hornby, G., and Linden, D. (2004). Evolutionary Antenna Design for a NASA Spacecraft. In O'Reilly, U.-M., Yu, T., Riolo, R. L., and Worzel, B., editors, *Genetic Programming Theory and Practice II*, chapter 18, pp 301–315. Springer, Ann Arbor.

[Lohn et al., 2008] Lohn, J. D., Hornby, G., and Linden, D. S. (2008). Human-competitive evolved antennas. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 22(3):235–247. Cambridge University Press.

[Lones and Tyrrell, 2001] Lones, M. A. and Tyrrell, A. M. (2001). Enzyme genetic programming. In *IEEE 2001 Congress on Evolutionary Computation, CEC'01*, pp 1183–1190. IEEE.

[Lones and Tyrrell, 2002] Lones, M. A. and Tyrrell, A. M. (2002). Biomimetic representation with genetic programming enzyme. *Genetic Programming and Evolvable Machines*, 3(2):193–217. Springer.

[Lopes and Costa, 2011a] Lopes, R. L. and Costa, E. (2011a). ReNCoDe: a regulatory network computational device. In *EuroGP'11: Proceedings of the 14th European Conference on Genetic Programming*, pp 142–153. Springer.

[Lopes and Costa, 2011b] Lopes, R. L. and Costa, E. (2011b). The squares problem and a neutrality analysis with ReNCoDe. In Proceedings of the 15th Portuguese Conference on Progress in Artificial Intelligence, EPIA'11, pp 182–195. Springer.

[Lopes and Costa, 2011c] Lopes, R. L. and Costa, E. (2011c). Using feedback in a regulatory network computational device. In *GECCO '11: Proceedings of the 13th annual Conference on Genetic and Evolutionary Computation*, pp 1499–1506. ACM.

[Lopes and Costa, 2012] Lopes, R. L. and Costa, E. (2012). The Regulatory Network Computational Device. *Genetic Programming and Evolvable Machines*, 13(3):339–375. Springer.

[Lopes and Costa, 2013a] Lopes, R. L. and Costa, E. (2013a). Evolving an Harmonic Number Generator with ReNCoDe. In *Proceedings of the 16th Portuguese Conference on Artificial Intelligence, EPIA'13*, pp 102–113. Springer.

[Lopes and Costa, 2013b] Lopes, R. L. and Costa, E. (2013b). Genetic programming with genetic regulatory networks. In *GECCO '13: Proceedings of the 15th annual Conference on Genetic and Evolutionary Computation*, pp 965–972. ACM.

[Marbach et al., 2007] Marbach, D., Mattiussi, C., and Floreano, D. (2007). Bio-mimetic Evolutionary Reverse Engineering of Genetic Regulatory Networks. *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, 4447:155–165. Springer.

[McDermott et al., 2012] McDermott, J., White, D. R., Luke, S., Manzoni, L., Castelli, M., Vanneschi, L., Jaśkowski, W., Krawiec, K., Harper, R., and De Jong, K. (2012). Genetic programming needs better benchmarks. *GECCO'12: Proceedings of the 14th annual Conference on Genetic and Evolutionary Computation*, pp 791–798. ACM.

[McKay et al., 2010] McKay, R. I., Hoai, N. X., Whigham, P. A., Shan, Y., and O'Neill, M. (2010). Grammar-based Genetic Programming: a survey. *Genetic Programming and Evolvable Machines*, 11(3-4):365–396. Springer.

[Miller, 1999] Miller, J. F. (1999). An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. *EuroGP'99: Proceedings of the Second European Conference on Genetic Programming*, 2:1135–1142. Springer.

[Miller, 2011] Miller, J. F. (2011). *Cartesian Genetic Programming*. Springer.

[Miller and Thomson, 2000] Miller, J. F. and Thomson, P. (2000). Cartesian Genetic Programming. In *EuroGP'00: Proceedings of the Third European Conference on Genetic Programming*, pp 121–132. Springer.

[Miller et al., 1998] Miller, J. F., Thomson, P., and Fogarty, T. C. (1998). Designing electronic circuits using evolutionary algorithms. In Quagliarella, D., Périaux, J., Poloni, C., and Winter, G., editors, *Arithmetic circuits: A case study*, pp 105–131. John Wiley & Sons, Chichester.

[Murphy et al., 2012] Murphy, E., Nicolau, M., Hemberg, E., O'Neill, M., and Brabazon, A. (2012). Differential Gene Expression with Tree-Adjunct Grammars. *PPSN'12: Proceedings of the 12th Conference on Parallel Problem Solving from Nature*, pp 377–386. Springer.

[Nicolau et al., 2012] Nicolau, M., O'Neill, M., and Brabazon, A. (2012). Applying Genetic Regulatory Networks to Index Trading. *PPSN'12: Proceedings of the 12th Conference on Parallel Problem Solving from Nature*, pp 428–437. Springer.

[Nicolau and Schoenauer, 2009] Nicolau, M. and Schoenauer, M. (2009). Evolving specific network statistical properties using a gene regulatory network model. *GECCO '09: Proceedings of the 11th annual Conference on Genetic and Evolutionary Computation*, pp 723–730. ACM.

[Nicolau et al., 2010] Nicolau, M., Schoenauer, M., and Banzhaf, W. (2010). Evolving Genes to Balance a Pole. EuroGP'10: Proceedings of the 13th European Conference on Genetic Programming, pp 196–207. Springer.

[Noman and Iba, 2007] Noman, N. and Iba, H. (2007). Inferring Gene Regulatory Networks using Differential Evolution with Local Search Heuristics. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB*, 4(4):634–647. IEEE.

[Nordin, 1994] Nordin, P. (1994). A compiling genetic programming system that directly manipulates the machine code. *Advances in Genetic Programming*, pp 311–331. MIT Press.

[Nüsslein-Volhard, 2006] Nüsslein-Volhard, C. (2006). *Coming to Life: How Genes Drive Development*. Kales Press.

[O'Neill and Ryan, 2003] O'Neill, M. and Ryan, C. (2003). *Grammatical Evolution*. Evolutionary Automatic Programming in an Arbitrary Language. Springer.

[O'Neill et al., 2003] O'Neill, M., Ryan, C., Keijzer, M., and Cattolico, M. (2003). Crossover in grammatical evolution. *Genetic Programming and Evolvable Machines*, 4(1):67–93. Springer.

[Poli, 1998] Poli, R. (1998). Efficient Evolution of Parallel Binary Multipliers and Continuous Symbolic Regression Expressions with Sub-Machine-Code GP. Technical report, University of Birmingham, School of Computer Science.

[Poli et al., 2011] Poli, R., Salvaris, M., and Cinel, C. (2011). Evolution of a Brain-Computer Interface Mouse via Genetic Programming. In Silva, S., Foster, J. A., Nicolau, M., Giacobini, M., and Machado, P., editors, *EuroGP'11: Proceedings of the 14th European Conference on Genetic Programming*, volume 6621 of *LNCS*, pp 203–214, Turin, Italy. Springer.

[R Development Core Team, 2008] R Development Core Team (2008). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

[Russell and Norvig, 2010] Russell, S. J. and Norvig, P. (2010). *Artificial Intelligence*. A Modern Approach. Prentice Hall.

[Sakamoto and Iba, 2001] Sakamoto, E. and Iba, H. (2001). Inferring a system of differential equations for a gene regulatory network by using genetic programming. In *IEEE 2001 Congress on Evolutionary Computation, CEC'01*, pp 720–726. IEEE.

[Schlitt and Brazma, 2007] Schlitt, T. and Brazma, A. (2007). Current approaches to gene regulatory network modelling. *BMC Bioinformatics*, 8(Suppl 6):S9. BioMed Central.

[Schwefel, 1995] Schwefel, H.-P. (1995). *Evolution and optimum seeking*. Wiley-Interscience.

[Sen, 1998] Sen, P. (1998). Designing Digital Adder Structures Through the Use of Genetic Programming. In Koza, J. R., editor, *Genetic Algorithms and Genetic Programming*, pp 167–176. Stanford Bookstore, Stanford, California, 94305-3079 USA.

[Shubin, 2009] Shubin, N. (2009). *Your Inner Fish*. The amazing discovery of our 375-million-year-old ancestor. Penguin Books Ltd.

[Silva and Costa, 2009] Silva, S. and Costa, E. (2009). Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories. *Genetic Programming and Evolvable Machines*, 10(2):141–179. Springer.

[Sousa and Costa, 2011] Sousa, J. A. B. and Costa, E. (2011). Designing an Epigenetic Approach in Artificial Life: The EpiAL Model. In *Agents and Artificial Intelligence*, pp 78–90. Springer Berlin, Heidelberg.

[Spector and Stoffel, 1996] Spector, L. and Stoffel, K. (1996). Ontogenetic programming. In *Genetic Programming '96: Proceedings of the 1st annual Conference on Genetic Programming*. ACM.

[Streeter, 2001] Streeter, M. J. (2001). *Automated Discovery of Numerical Approximation Formulae via Genetic Programming*. PhD thesis, Worcester Polytechnic Institute, Worcester Polytechnic Institute.

[Streichert et al., 2004] Streichert, F., Planatscher, H., Spieth, C., and Ulmer, H. (2004). Comparing Genetic Programming and Evolution Strategies on Inferring Gene Regulatory Networks. *GECCO'04: Proceedings of the 6th annual Conference on Genetic and Evolutionary Computation*, Volume 3102, pp 471–480 Springer.

[Tanev and Yuta, 2008] Tanev, I. and Yuta, K. (2008). Epigenetic programming: Genetic programming incorporating epigenetic learning through modification of histones. *Information Sciences: an International Journal*, 178(23). Elsevier.

[Teichmann and Babu, 2004] Teichmann, S. A. and Babu, M. M. (2004). Gene regulatory network growth by duplication. *Nature Genetics*, 36(5):492–496. Macmillan Publishers Limited.

[Trefzer et al., 2011] Trefzer, M. A., Kuyucu, T., Miller, J. F., and Tyrrell, A. M. (2011). On the Advantages of Variable Length GRNs for the Evolution of Multicellular Developmental Systems. *IEEE Transactions on Evolutionary Computation*, 17(1):100–121. IEEE.

[Turner et al., 2013] Turner, A. P., Lones, M. A., Fuente, L. A., Stepney, S., Caves, L. S. D., and Tyrrell, A. (2013). The artificial epigenetic network. In *2013 IEEE International Conference on Evolvable Systems (ICES)*, pp 66–72. IEEE.

[Vassilev and Miller, 2000] Vassilev, V. K. and Miller, J. F. (2000). Embedding landscape neutrality to build a bridge from the conventional to a more efficient three-bit multiplier circuit. *GECCO'00: Proceedings of the Genetic and Evolutionary Computation Conference*, page 539. Morgan Kaufmann Publishers.

[Vassilev et al., 1999] Vassilev, V. K., Miller, J. F., and Fogarty, T. C. (1999). On the nature of two-bit multiplier landscapes. In *First NASA/DoD Workshop on Evolvable Hardware*, pp 36–45. IEEE Computer Society.

[Vijesh, 2013] Vijesh, N. (2013). Modeling of gene regulatory networks: A review. *Journal of Biomedical Science and Engineering*, 06(02):223–231. Academic Research Publishing.

[Walker and Miller, 2004a] Walker, J. and Miller, J. (2004a). Evolution and Acquisition of Modules in Cartesian Genetic Programming. In Keijzer, M., O'Reilly, U.-M., Lucas, S., Costa, E., and Soule, T., editors, *Lecture Notes in Computer Science*, pp 187–197. Springer.

[Walker and Miller, 2004b] Walker, J. A. and Miller, J. F. (2004b). The Automatic Acquisition, Evolution and Reuse of Modules in Cartesian Genetic Programming. *IEEE Transactions on Evolutionary Computation*, 12(4):397–417. IEEE.

[Walker and Miller, 2005a] Walker, J. A. and Miller, J. F. (2005a). Improving the Evolvability of Digital Multipliers Using Embedded Cartesian Genetic Programming and Product Reduction. Evolvable Systems: From Biology to Hardware. Lecture Notes in Computer Science, Volume 3637, pp 131-142. Springer.

[Walker and Miller, 2005b] Walker, J. A. and Miller, J. F. (2005b). Investigating the performance of module acquisition in cartesian genetic programming. *GECCO'05: Proceedings of the 7th annual Conference on Genetic and Evolutionary Computation*, pp 1649–1656. ACM.

[White et al., 2012] White, D. R., McDermott, J., Castelli, M., Manzoni, L., Goldman, B. W., Kronberger, G., Jaśkowski, W., O'Reilly, U.-M., and Luke, S. (2012). Better GP benchmarks: community survey results and proposals. *Genetic Programming and Evolvable Machines*, 14(1), pp 3-29. Springer.

[Whitley et al., 1993] Whitley, D., Dominic, S., Das, R., and Anderson, C. W. (1993). Genetic reinforcement learning for neurocontrol problems. *Machine Learning*, 13(2):259–284. Springer.

[Wong and Leung, 1995] Wong, M. L. and Leung, K. S. (1995). An induction system that learns programs in different programming languages using genetic programming and logic grammars. In *Tools with Artificial Intelligence, 1995. Proceedings., Seventh International Conference on*, pp 380–387. IEEE Computer Society.

[Wong and Leung, 1996] Wong, M. L. and Leung, K. S. (1996). Evolving Recursive Functions for the Even-Parity Problem Using Genetic Programming. In Angeline, P. J. and K E Kinnear, J., editors, *Advances in Genetic Programming*, pp 221–240. MIT Press.

[Wong and Mun, 2005] Wong, M. L. and Mun, T. (2005). Evolving Recursive Programs by Using Adaptive Grammar Based Genetic Programming. *Genetic Programming and Evolvable Machines*, 6(4):421–455. Springer.

Appendices

A

Results Data

A.1 ARN Design Alternatives

Table A.1: Summary of the number of evaluations necessary to find an optimal solution for each experiment. The results are split by problem (group1) and the labels refer the initialisation method (*dm* or *rnd*), the size (using the corresponding DM event number from 5 to 9), and if genes overlap or not (*T* or *F*).

group1	group2	n	mean	sd	min	se
harmonic	dm5T	87	191741	223816	2100	23996
	dm6F	9	508656	349203	74300	116401
	dm6T	94	122181	175782	900	18131
	dm7F	96	156253	216054	1300	22051
	dm7T	99	78432	112175	1700	11274
	dm8F	98	87309	116153	1200	11733
	dm8T	100	94243	132647	2200	13265
	dm9F	100	97575	131345	1700	13135
	dm9T	100	100882	125924	2100	12592
	rnd5T	60	278952	247265	11100	31922
	rnd6F	1	139500	-	139500	-
	rnd6T	89	168591	209128	1600	22168
	rnd7F	88	159231	170690	7200	18196
	rnd7T	97	84428	136959	1200	13906
	rnd8F	100	75877	111795	2400	11179

group1	group2	n	mean	sd	min	se
	rnd8T	100	75364	89049	400	8905
	rnd9F	100	96587	76640	2200	7664
	rnd9T	97	144086	139594	9100	14174
pendulum	dm5T	79	179547	255981	1200	28800
	dm6F	89	149474	202240	1000	21437
	dm6T	98	96476	164282	1000	16595
	dm7F	97	47960	106445	400	10808
	dm7T	98	83473	169394	400	17111
	dm8F	99	46912	102801	200	10332
	dm8T	98	31938	81535	700	8236
	dm9F	99	32343	92714	400	9318
	dm9T	98	34514	69219	400	6992
	rnd5T	100	63906	172779	300	17278
	rnd6F	100	20247	39836	500	3984
	rnd6T	100	14889	37210	600	3721
	rnd7F	100	6145	5476	600	548
	rnd7T	100	8345	8332	100	833
	rnd8F	100	27203	33610	1300	3361
	rnd8T	99	68134	104728	1100	10526
	rnd9F	96	220117	189965	2700	19388
	rnd9T	84	298185	252157	11200	27513
polinomial	dm5T	78	246759	253963	900	28756
	dm6F	3	329333	33780	296400	19503
	dm6T	85	162938	204734	1300	22207
	dm7F	92	150627	198799	2800	20726
	dm7T	80	195655	229185	1900	25624
	dm8F	88	136610	186692	1900	19901
	dm8T	48	192200	266103	500	38409
	dm9F	38	224224	307084	2100	49816
	dm9T	27	177381	276550	2300	53222
	rnd5T	92	147192	180510	5900	18819
	rnd6F	4	171075	119946	39900	59973

group1	group2	n	mean	sd	min	se
	rnd6T	99	87134	170443	1100	17130
	rnd7F	99	71065	122588	400	12321
	rnd7T	100	111402	204648	600	20465
	rnd8F	98	89632	156127	300	15771
	rnd8T	77	240995	243760	2900	27779
	rnd9F	20	541315	307080	79200	68665
	rnd9T	1	234800	-	234800	-
santafetrail	dm5T	60	194322	248017	4400	32019
	dm6F	13	475346	272753	98700	75648
	dm6T	75	216364	253941	1800	29323
	dm7F	78	153764	209470	2100	23718
	dm7T	67	128975	190983	600	23332
	dm8F	61	183038	235093	2600	30101
	dm8T	50	180538	176333	1300	24937
	dm9F	47	219617	256043	6700	37348
	dm9T	39	208095	220295	11400	35275
	rnd5T	76	265547	257747	5900	29566
	rnd6F	22	404250	298110	5100	63557
	rnd6T	94	134336	188945	2700	19488
	rnd7F	94	101879	128448	4200	13248
	rnd7T	93	121020	191712	3600	19880
	rnd8F	99	133993	175230	1700	17611
	rnd8T	90	166123	170124	6400	17933
	rnd9F	53	459532	248736	102700	34166
	rnd9T	2	166200	49073	131500	34700

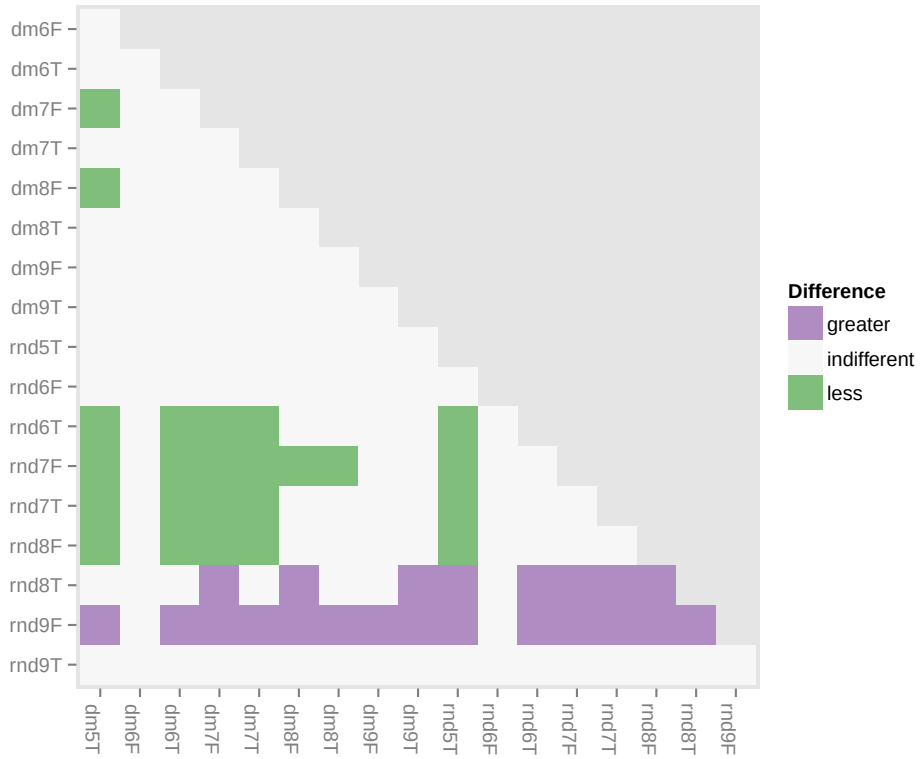


Figure A.1: Mann-Whitney-Wilcoxon test results for the design alternatives with the polinomial regression problem, showing *less* if the number of evaluations of the experiment in the *y* axis is significantly less than the one on the *x* axis, with *greater* otherwise, or *not significant* if the difference is not statistically significant. The labels refer the initialisation method (*dm* or *rnd*), the size (using the corresponding DM event number from 5 to 9), and if genes overlap or not (*T* or *F*).

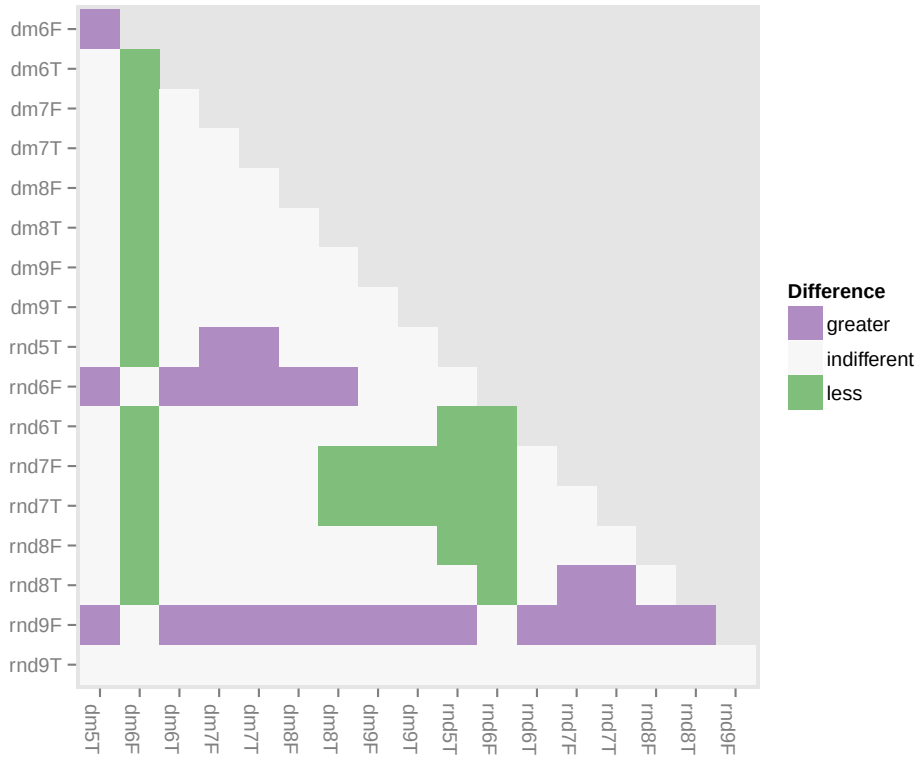


Figure A.2: Mann-Whitney-Wilcoxon test results for the design alternatives with the Santa Fe ant trail problem, showing *less* if the number of evaluations of the experiment in the *y* axis is significantly less than the one on the *x* axis, with *greater* otherwise, or *not significant* if the difference is not statistically significant. The labels refer the initialisation method (*dm* or *rnd*), the size (using the corresponding DM event number from 5 to 9), and if genes overlap or not (*T* or *F*).

Listing A.1: Results of the one-tailed pairwise comparisons for the harmonic regression problem, using the alternative hypothesis that the row is less than the column.

	dm5T	dm6F	dm6T	dm7F	dm7T	dm8F	dm8T	dm9F	dm9T	rnd5T	rnd6F	rnd6T	rnd7F	rnd7T	rnd8F	rnd8T	rnd9F
dm6F	1.0000	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
dm6T	1.0000	0.0210	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
dm7F	1.0000	0.0935	1.0000	-	-	-	-	-	-	-	-	-	-	-	-	-	-
dm7T	0.0519	0.0020	1.0000	1.0000	-	-	-	-	-	-	-	-	-	-	-	-	-
dm8F	0.1787	0.0040	1.0000	1.0000	1.0000	-	-	-	-	-	-	-	-	-	-	-	-
dm8T	1.0000	0.0049	1.0000	1.0000	1.0000	1.0000	-	-	-	-	-	-	-	-	-	-	-
dm9F	0.6350	0.0069	1.0000	1.0000	1.0000	1.0000	1.0000	-	-	-	-	-	-	-	-	-	-
dm9T	1.0000	0.0069	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	-	-	-	-	-	-	-	-	-
rnd5T	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	-	-	-	-	-	-	-	-
rnd6F	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	-	-	-	-	-	-	-
rnd6T	1.0000	0.1013	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.0490	1.0000	-	-	-	-	-	-
rnd7F	1.0000	0.0958	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.0814	1.0000	1.0000	-	-	-	-	-
rnd7T	0.0787	0.0022	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.7e-08	1.0000	0.0187	0.0133	-	-	-	-
rnd8F	0.3036	0.0012	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	2.4e-09	1.0000	0.0348	0.0310	1.0000	-	-	-
rnd8T	0.2908	0.0027	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.3e-08	1.0000	0.1073	0.0318	1.0000	1.0000	-	-
rnd9F	1.0000	0.0081	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	4.5e-05	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	-
rnd9T	1.0000	0.0840	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.0268	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

Listing A.2: Results of the one-tailed pairwise comparisons for the harmonic regression problem, using the alternative hypothesis that the row is greater than the column.

	dm5T	dm6F	dm6T	dm7F	dm7T	dm8F	dm8T	dm9F	dm9T	rnd5T	rnd6F	rnd6T	rnd7F	rnd7T	rnd8F	rnd8T	rnd9F
dm6F	0.27107	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
dm6T	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
dm7F	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-	-	-
dm7T	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-	-
dm8F	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-
dm8T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-
dm9F	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-
dm9T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-
rnd5T	0.26073	1.00000	1.8e-05	0.00334	7.1e-09	6.3e-08	2.5e-07	7.9e-07	3.4e-06	-	-	-	-	-	-	-	-
rnd6F	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-
rnd6T	1.00000	1.00000	1.00000	1.00000	0.01664	0.06481	0.37693	0.31233	1.00000	1.00000	1.00000	-	-	-	-	-	-
rnd7F	1.00000	1.00000	0.74871	1.00000	0.00702	0.02255	0.12230	0.15644	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-
rnd7T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-
rnd8F	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-
rnd8T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-
rnd9F	1.00000	1.00000	1.00000	1.00000	0.01097	0.10203	0.57078	0.83938	1.00000	1.00000	1.00000	1.00000	1.00000	0.00593	0.00690	0.16235	-
rnd9T	1.00000	1.00000	0.19336	1.00000	0.00023	0.00182	0.01661	0.02001	0.31694	1.00000	1.00000	1.00000	1.00000	0.00017	0.00030	0.00219	1.00000

Listing A.3: Results of the one-tailed pairwise comparisons for the pendulum problem, using the alternative hypothesis that the row is less than the column.

```
1         dm5T    dm6F    dm6T    dm7F    dm7T    dm8F    dm8T    dm9F    dm9T    rnd5T   rnd6F   rnd6T   rnd7F   rnd7T   rnd8F   rnd8T   rnd9F
2  dm6F   1.00000 -       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
3  dm6T   0.84266 1.00000 -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
4  dm7F   1.8e-06 2.4e-06 0.00815 -       -       -       -       -       -       -       -       -       -       -       -       -       -
5  dm7T   0.00111 0.00237 0.93107 1.00000 -       -       -       -       -       -       -       -       -       -       -       -       -
6  dm8F   1.7e-07 1.1e-07 0.00071 1.00000 1.00000 -       -       -       -       -       -       -       -       -       -       -       -
7  dm8T   1.7e-09 9.9e-10 1.7e-05 1.00000 1.00000 1.00000 -       -       -       -       -       -       -       -       -       -       -
8  dm9F   1.2e-11 4.9e-12 2.0e-08 0.16991 0.01433 1.00000 1.00000 -       -       -       -       -       -       -       -       -       -
9  dm9T   2.8e-09 1.5e-09 1.1e-05 1.00000 0.48238 1.00000 1.00000 1.00000 -       -       -       -       -       -       -       -       -
10 rnd5T  3.3e-06 5.2e-06 0.01901 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -       -       -       -       -       -
11 rnd6F  1.2e-10 1.8e-10 3.2e-06 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -       -       -       -       -
12 rnd6T  1.2e-12 5.5e-13 1.0e-08 1.00000 0.06234 1.00000 1.00000 1.00000 1.00000 0.18412 1.00000 -       -       -       -       -       -
13 rnd7F  < 2e-16 < 2e-16 1.6e-14 0.00081 3.4e-05 0.15385 0.30886 1.00000 1.00000 2.8e-06 0.02680 0.69065 -       -       -       -       -
14 rnd7T  1.9e-15 2.6e-15 3.6e-11 0.20338 0.00615 1.00000 1.00000 1.00000 1.00000 0.00423 1.00000 1.00000 1.00000 -       -       -       -
15 rnd8F  3.1e-05 6.4e-05 0.51461 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -
16 rnd8T  1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -
17 rnd9F  1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -
18 rnd9T  1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000
```

Listing A.4: Results of the one-tailed pairwise comparisons for the pendulum problem, using the alternative hypothesis that the row is greater than the column.

```
1         dm5T    dm6F    dm6T    dm7F    dm7T    dm8F    dm8T    dm9F    dm9T    rnd5T   rnd6F   rnd6T   rnd7F   rnd7T   rnd8F   rnd8T   rnd9F
2  dm6F   1.00000 -       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
3  dm6T   1.00000 1.00000 -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
4  dm7F   1.00000 1.00000 1.00000 -       -       -       -       -       -       -       -       -       -       -       -       -       -
5  dm7T   1.00000 1.00000 1.00000 1.00000 -       -       -       -       -       -       -       -       -       -       -       -       -
6  dm8F   1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -       -       -       -       -       -       -       -       -       -
7  dm8T   1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -       -       -       -       -       -       -       -       -
8  dm9F   1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -       -       -       -       -       -       -       -
9  dm9T   1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -       -       -       -       -       -       -
10 rnd5T  1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 0.01979 0.75053 -       -       -       -       -       -       -
11 rnd6F  1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -       -       -       -       -
12 rnd6T  1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -       -       -       -
13 rnd7F  1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -       -       -
14 rnd7T  1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -       -
15 rnd8F  1.00000 1.00000 1.00000 0.77040 1.00000 0.10057 0.00267 1.9e-06 0.00110 1.00000 0.00203 1.2e-07 1.4e-15 4.0e-11 -       -       -
16 rnd8T  1.00000 1.00000 1.00000 2.2e-07 0.00241 7.5e-08 1.1e-10 6.6e-14 1.2e-09 6.9e-08 8.6e-14 < 2e-16 < 2e-16 < 2e-16 2.8e-07 -       -
17 rnd9F  0.02751 0.00474 9.1e-10 < 2e-16 4.8e-13 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 1.5e-13 -
18 rnd9T  0.00039 1.6e-05 7.5e-12 < 2e-16 9.2e-15 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 1.1e-14 1.00000
```

Listing A.5: Results of the one-tailed pairwise comparisons for the polinomial regression problem, using the alternative hypothesis that the row is less than the column.

```
1         dm5T    dm6F    dm6T    dm7F    dm7T    dm8F    dm8T    dm9F    dm9T    rnd5T   rnd6F   rnd6T   rnd7F   rnd7T   rnd8F   rnd8T   rnd9F
2  dm6F   1.00000 -       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
3  dm6T   1.00000 1.00000 -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
4  dm7F   0.21590 1.00000 1.00000 -       -       -       -       -       -       -       -       -       -       -       -       -       -
5  dm7T   1.00000 1.00000 1.00000 1.00000 -       -       -       -       -       -       -       -       -       -       -       -       -
6  dm8F   0.00996 1.00000 1.00000 1.00000 1.00000 -       -       -       -       -       -       -       -       -       -       -       -
7  dm8T   1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -       -       -       -       -       -       -       -       -
8  dm9F   1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -       -       -       -       -       -       -       -
9  dm9T   0.52183 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -       -       -       -       -       -       -
10 rnd5T  0.70296 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -       -       -       -       -       -
11 rnd6F  1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -       -       -       -       -
12 rnd6T  1.3e-08 1.00000 0.00057 0.00133 0.00056 1.00000 1.00000 1.00000 1.00000 6.2e-06 1.00000 -       -       -       -       -       -
13 rnd7F  1.7e-09 0.64511 0.00011 0.00024 6.6e-05 0.23855 0.39519 0.45915 1.00000 2.1e-06 1.00000 1.00000 -       -       -       -       -
14 rnd7T  7.5e-07 1.00000 0.00653 0.01251 0.00211 0.73835 0.87400 0.73800 1.00000 0.00046 1.00000 1.00000 1.00000 -       -       -       -
15 rnd8F  2.0e-07 0.82272 0.00620 0.01089 0.00126 1.00000 1.00000 1.00000 1.00000 0.00059 1.00000 1.00000 1.00000 1.00000 -       -       -
16 rnd8T  1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -
17 rnd9F  1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -
18 rnd9T  1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000
```

Listing A.6: Results of the one-tailed pairwise comparisons for the polinomial regression problem, using the alternative hypothesis that the row is greater than the column.

```
1         dm5T    dm6F    dm6T    dm7F    dm7T    dm8F    dm8T    dm9F    dm9T    rnd5T   rnd6F   rnd6T   rnd7F   rnd7T   rnd8F   rnd8T   rnd9F
2  dm6F   1.00000 -       -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
3  dm6T   1.00000 1.00000 -       -       -       -       -       -       -       -       -       -       -       -       -       -       -
4  dm7F   1.00000 1.00000 1.00000 -       -       -       -       -       -       -       -       -       -       -       -       -       -
5  dm7T   1.00000 1.00000 1.00000 1.00000 -       -       -       -       -       -       -       -       -       -       -       -       -
6  dm8F   1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -       -       -       -       -       -       -       -       -       -
7  dm8T   1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -       -       -       -       -       -       -       -       -
8  dm9F   1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -       -       -       -       -       -       -       -
9  dm9T   1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -       -       -       -       -       -       -
10 rnd5T  1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -       -       -       -       -       -
11 rnd6F  1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -       -       -       -       -
12 rnd6T  1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -       -       -       -
13 rnd7F  1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -       -       -
14 rnd7T  1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -       -
15 rnd8F  1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 -       -       -
16 rnd8T  1.00000 1.00000 0.46510 0.09390 1.00000 0.00601 0.83423 1.00000 0.40135 0.17824 1.00000 8.4e-09 1.2e-09 3.0e-07 5.8e-08 -       -
17 rnd9F  0.00761 1.00000 2.9e-05 8.5e-06 0.00036 4.9e-06 0.00029 0.01418 0.00122 5.2e-06 1.00000 5.6e-08 1.8e-08 3.8e-07 1.2e-07 0.00879 -
18 rnd9T  1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000
```

Listing A.7: Results of the one-tailed pairwise comparisons for the Santa Fe trail problem, using the alternative hypothesis that the row is greater than the column.

	dm5T	dm6F	dm6T	dm7F	dm7T	dm8F	dm8T	dm9F	dm9T	rnd5T	rnd6F	rnd6T	rnd7F	rnd7T	rnd8F	rnd8T	rnd9F
dm6F	1.00000	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
dm6T	1.00000	0.05410	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
dm7F	1.00000	0.00128	1.00000	-	-	-	-	-	-	-	-	-	-	-	-	-	-
dm7T	1.00000	0.00025	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-	-
dm8F	1.00000	0.00761	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-
dm8T	1.00000	0.01579	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-
dm9F	1.00000	0.05320	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-
dm9T	1.00000	0.04624	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-
rnd5T	1.00000	0.37071	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-
rnd6F	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-
rnd6T	1.00000	0.00027	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	0.00073	0.00110	-	-	-	-	-	-
rnd7F	1.00000	3.4e-05	1.00000	1.00000	1.00000	1.00000	0.38221	0.15315	0.14370	1.2e-05	0.00015	1.00000	-	-	-	-	-
rnd7T	1.00000	0.00014	0.96770	1.00000	1.00000	1.00000	0.19171	0.06556	0.05927	1.3e-05	0.00063	1.00000	1.00000	-	-	-	-
rnd8F	1.00000	0.00025	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	0.00126	0.00240	1.00000	1.00000	1.00000	-	-	-
rnd8T	1.00000	0.00182	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	0.73829	0.03050	1.00000	1.00000	1.00000	1.00000	-	-
rnd9F	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-
rnd9T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000

Listing A.8: Results of the one-tailed pairwise comparisons for the Santa Fe trail problem, using the alternative hypothesis that the row is less than the column.

	dm5T	dm6F	dm6T	dm7F	dm7T	dm8F	dm8T	dm9F	dm9T	rnd5T	rnd6F	rnd6T	rnd7F	rnd7T	rnd8F	rnd8T	rnd9F
dm6F	0.03234	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
dm6T	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
dm7F	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-	-	-
dm7T	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-	-
dm8F	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-
dm8T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-
dm9F	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-
dm9T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-
rnd5T	0.64963	1.00000	1.00000	0.00949	0.00295	0.50626	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-
rnd6F	0.10613	1.00000	0.29450	0.00894	0.00253	0.06226	0.16806	0.59965	0.50793	1.00000	-	-	-	-	-	-	-
rnd6T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-
rnd7F	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-
rnd7T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-
rnd8F	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-
rnd8T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	0.08182	0.02532	1.00000	-	-
rnd9F	9.5e-07	1.00000	2.8e-06	2.6e-10	6.3e-12	1.1e-07	6.3e-07	2.6e-05	2.8e-05	0.00038	1.00000	3.5e-13	8.6e-16	2.0e-13	3.8e-13	2.9e-10	-
rnd9T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000

Table A.2: Summary of the generalisation results for each experiment in the harmonic and pendulum problems. The results are split by problem (group1) and the labels refer the initialisation method (*dm* or *rnd*), the size (using the corresponding DM event number from 5 to 9), and if genes overlap or not (*T* or *F*).

group1	group2	n	mean	sd	median	min	max
harmonic	dm5T	87	0.0122	0.0102	0.0085	0.0022	0.0546
	dm6F	9	0.0140	0.0081	0.0175	0.0022	0.0215
	dm6T	94	0.0131	0.0089	0.0107	0.0022	0.0626
	dm7F	96	0.0114	0.0078	0.0093	0.0022	0.0403
	dm7T	99	0.0122	0.0092	0.0093	0.0010	0.0563
	dm8F	98	0.0134	0.0094	0.0104	0.0022	0.0578
	dm8T	100	0.0152	0.0111	0.0128	0.0021	0.0678
	dm9F	100	0.0129	0.0086	0.0104	0.0022	0.0418
	dm9T	99	0.0139	0.0110	0.0101	0.0010	0.0616
	rnd5T	60	0.0109	0.0107	0.0073	0.0020	0.0695
	rnd6F	1	0.0054	-	0.0054	0.0054	0.0054
	rnd6T	89	0.0089	0.0075	0.0068	0.0010	0.0482
	rnd7F	88	0.0103	0.0085	0.0068	0.0020	0.0367
	rnd7T	97	0.0098	0.0073	0.0067	0.0010	0.0400
	rnd8F	100	0.0105	0.0081	0.0080	0.0015	0.0449
	rnd8T	100	0.0125	0.0080	0.0107	0.0010	0.0408
	rnd9F	100	0.0138	0.0093	0.0107	0.0022	0.0467
	rnd9T	92	0.0136	0.0083	0.0112	0.0026	0.0414
pendulum	dm5T	79	156.5063	77.0271	171.0000	0.0000	291.0000
	dm6F	89	139.1236	77.2524	171.0000	3.0000	291.0000
	dm6T	98	144.5918	88.9009	171.0000	1.0000	357.0000
	dm7F	97	142.1959	87.4940	171.0000	3.0000	359.0000
	dm7T	98	157.5918	83.4451	171.0000	0.0000	397.0000
	dm8F	99	156.9394	99.4830	163.0000	5.0000	495.0000
	dm8T	98	174.2959	102.2860	171.0000	2.0000	400.0000
	dm9F	99	160.1010	92.4362	163.0000	5.0000	365.0000
	dm9T	98	154.2959	104.5345	136.0000	0.0000	429.0000
	rnd5T	100	94.2200	70.5898	63.0000	5.0000	247.0000

group1	group2	n	mean	sd	median	min	max
	rnd6F	100	85.2400	65.4194	31.0000	31.0000	171.0000
	rnd6T	100	133.4700	76.1173	171.0000	17.0000	291.0000
	rnd7F	100	134.5900	68.7863	171.0000	31.0000	253.0000
	rnd7T	100	146.7000	67.9496	171.0000	0.0000	291.0000
	rnd8F	100	133.8300	82.2874	165.5000	1.0000	323.0000
	rnd8T	99	106.7172	81.2754	65.0000	1.0000	305.0000
	rnd9F	96	71.2917	74.6813	41.0000	1.0000	379.0000
	rnd9T	84	60.7143	70.9098	35.5000	1.0000	350.0000

Figure A.3: The program evolved to control the inverted pendulum. The arithmetic operators were used and the four input variables correspond respectively to $x, \dot{x}, \theta, \dot{\theta}$.

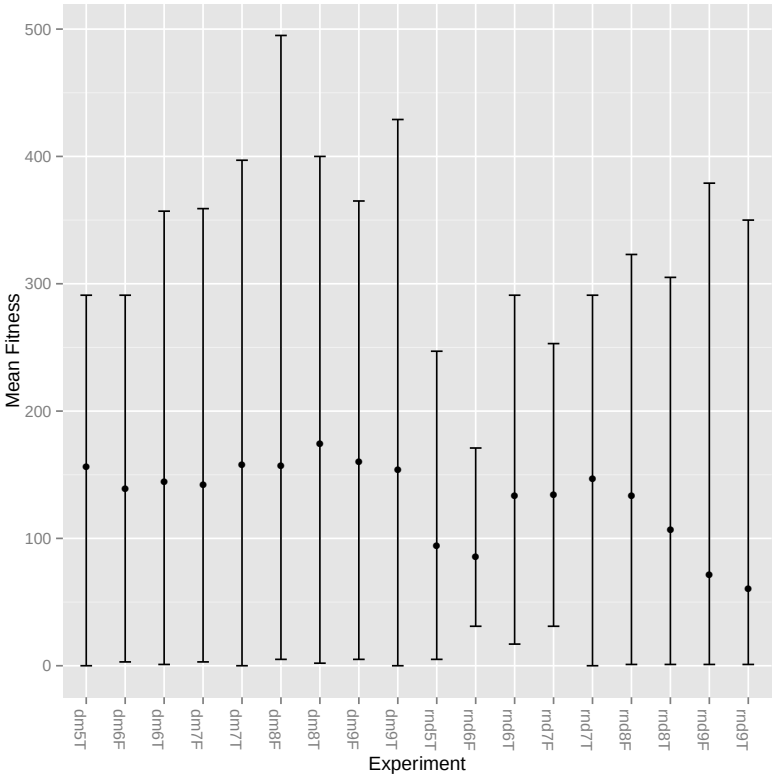


Figure A.4: Results of the generalisation task for the design alternatives with the inverted pendulum problem. The mean number of sucessful simulations (out of 625) is presented, with the minimum and maximum. The labels refer the initialisation method (*dm* or *rnd*), the size (using the corresponding DM event number from 5 to 9), and if genes overlap or not (*T* or *F*).

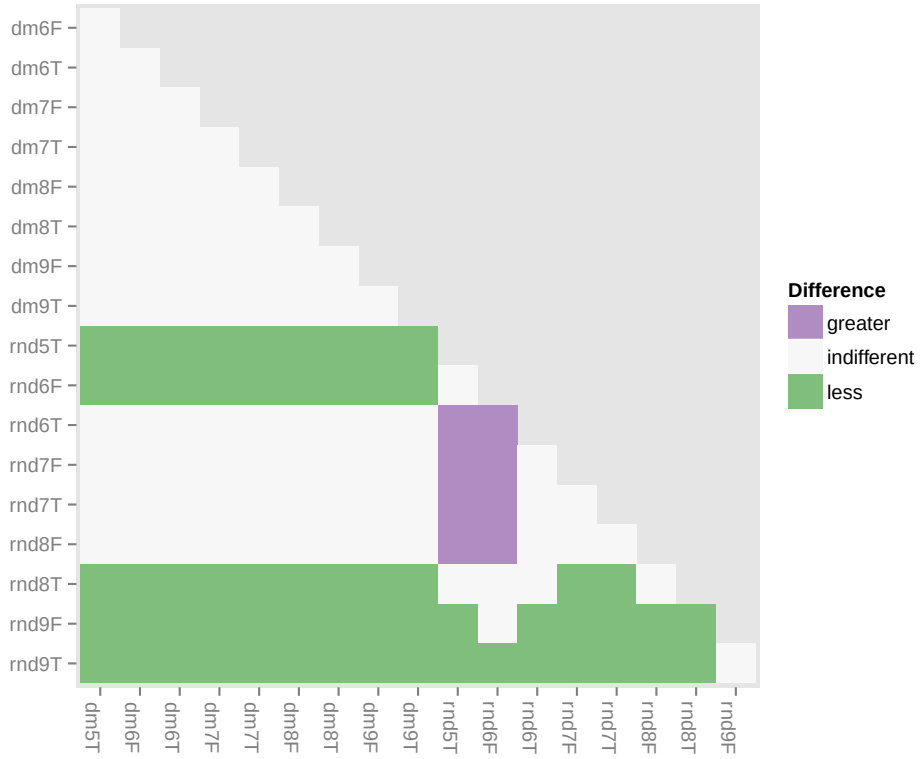


Figure A.5: Mann-Whitney-Wilcoxon test results of the generalisation task for the design alternatives, with the inverted pendulum problem, showing *less* if the number of evaluations of the experiment in the *y* axis is significantly less than the one on the *x* axis, with *greater* otherwise, or *not significant* if the difference is not statistically significant. The labels refer the initialisation method (*dm* or *rnd*), the size (using the corresponding DM event number from 5 to 9), and if genes overlap or not (*T* or *F*).

Listing A.9: Results of the one-tailed pairwise comparisons for the generalisation in the harmonic regression problem, using the alternative hypothesis that the row is less than the column.

	dm5T	dm6F	dm6T	dm7F	dm7T	dm8F	dm8T	dm9F	dm9T	rnd5T	rnd6F	rnd6T	rnd7F	rnd7T	rnd8F	rnd8T	rnd9F
dm6F	1.00000	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
dm6T	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
dm7F	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-	-	-
dm7T	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-	-
dm8F	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-
dm8T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-
dm9F	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-
dm9T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-
rnd5T	1.00000	1.00000	0.44113	1.00000	1.00000	0.14644	0.03553	1.00000	1.00000	-	-	-	-	-	-	-	-
rnd6F	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-
rnd6T	1.00000	1.00000	6.5e-05	0.22396	0.02388	1.5e-05	3.2e-06	0.00322	0.00307	1.00000	1.00000	-	-	-	-	-	-
rnd7F	1.00000	1.00000	0.06337	1.00000	1.00000	0.02930	0.00345	0.26124	0.30700	1.00000	1.00000	1.00000	-	-	-	-	-
rnd7T	1.00000	1.00000	0.04569	1.00000	1.00000	0.01729	0.00076	0.32015	0.36727	1.00000	1.00000	1.00000	1.00000	-	-	-	-
rnd8F	1.00000	1.00000	0.28859	1.00000	1.00000	0.09637	0.00568	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-
rnd8T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-
rnd9F	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-
rnd9T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000

Listing A.10: Results of the one-tailed pairwise comparisons for the generalisation in the harmonic regression problem, using the alternative hypothesis that the row is greater than the column.

	dm5T	dm6F	dm6T	dm7F	dm7T	dm8F	dm8T	dm9F	dm9T	rnd5T	rnd6F	rnd6T	rnd7F	rnd7T	rnd8F	rnd8T	rnd9F
dm6F	1.00000	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
dm6T	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
dm7F	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-	-	-
dm7T	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-	-
dm8F	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-
dm8T	0.39272	1.00000	1.00000	0.53268	0.73435	1.00000	-	-	-	-	-	-	-	-	-	-	-
dm9F	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-
dm9T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-
rnd5T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-
rnd6F	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-
rnd6T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-
rnd7F	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-
rnd7T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-
rnd8F	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-
rnd8T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	0.00265	0.35820	0.31153	1.00000	-	-
rnd9F	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	0.33828	1.00000	0.00015	0.04189	0.03480	0.21032	1.00000	-
rnd9T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	0.24313	1.00000	7.8e-05	0.02814	0.01396	0.11035	1.00000	1.00000

Listing A.11: Results of the one-tailed pairwise comparisons for the generalisation in the pendulum problem, using the alternative hypothesis that the row is less than the column.

	dm5T	dm6F	dm6T	dm7F	dm7T	dm8F	dm8T	dm9F	dm9T	rnd5T	rnd6F	rnd6T	rnd7F	rnd7T	rnd8F	rnd8T	rnd9F
dm6F	1.00000	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
dm6T	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
dm7F	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-	-	-
dm7T	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-	-
dm8F	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-
dm8T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-
dm9F	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-
dm9T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-
rnd5T	0.00044	0.00206	0.00411	0.00343	2.6e-06	3.6e-05	3.5e-06	5.5e-05	0.00205	-	-	-	-	-	-	-	-
rnd6F	8.6e-06	3.9e-05	0.00015	8.6e-05	1.8e-08	4.9e-07	4.1e-08	6.2e-07	7.4e-05	1.00000	-	-	-	-	-	-	-
rnd6T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-
rnd7F	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-
rnd7T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-
rnd8F	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-
rnd8T	0.01959	0.18047	0.20460	0.18118	0.00206	0.01048	0.00030	0.00459	0.08611	1.00000	1.00000	0.68895	0.34488	0.01187	1.00000	-	-
rnd9F	7.5e-09	7.9e-09	2.4e-07	2.6e-08	1.6e-11	7.3e-10	1.2e-11	9.4e-11	5.7e-08	0.14660	0.64299	4.2e-08	1.3e-09	1.0e-11	1.6e-06	0.05003	-
rnd9T	4.2e-09	2.8e-09	2.1e-08	2.6e-09	2.5e-12	4.9e-11	9.2e-13	2.8e-12	2.6e-09	0.01265	0.05265	6.2e-09	5.3e-10	6.3e-12	6.4e-08	0.00089	1.00000

Listing A.12: Results of the one-tailed pairwise comparisons for the generalisation in the pendulum problem, using the alternative hypothesis that the row is greater than the column.

	dm5T	dm6F	dm6T	dm7F	dm7T	dm8F	dm8T	dm9F	dm9T	rnd5T	rnd6F	rnd6T	rnd7F	rnd7T	rnd8F	rnd8T	rnd9F
dm6F	1.00000	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
dm6T	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
dm7F	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-	-	-
dm7T	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-	-
dm8F	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-	-
dm8T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-	-
dm9F	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-	-
dm9T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-	-
rnd5T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-	-
rnd6F	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-	-	-	-	-	-
rnd6T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	0.01033	0.00028	-	-	-	-	-	-
rnd7F	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	0.00129	1.3e-05	1.00000	-	-	-	-	-
rnd7T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.1e-05	4.7e-08	1.00000	1.00000	-	-	-	-
rnd8F	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	0.04181	0.00203	1.00000	1.00000	1.00000	-	-	-
rnd8T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-	-
rnd9F	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	-
rnd9T	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000

A.2 Genetic Operators

Table A.3: Summary of the number of evaluations necessary to find an optimal solution for each asexual operator in every problem. The results are split by problem and operator, where *JD, TD*, and *GCD* correspond respectively to the bitwise *junk* and *tranposon* (coupled with the *delete* operator), and to the *genecopy* and *genedelete* operators. The rates are represented by the first decimal place, that is, *14* stands for *0.1-0.4*, the order consistent with the operators labels.

Problem	Op.	Rates	Op.Size	n	mean	sd	median	min
harmonic	GCD	14	other	89	180190	201659	119200	3100
		23	other	94	148788	197714	70500	1200
		32	other	97	129692	178891	59400	1800
		41	other	97	150749	216362	49200	5500
		55	other	93	167513	204068	96400	1700
	JD	14	100	81	222394	238749	118000	6400
			200	77	250703	245534	167900	5600
			400	65	268578	251066	188700	800
			50	86	173967	200895	100350	600
		23	100	91	195074	210020	107200	1300
			200	85	219489	229231	130000	7900
			400	78	175724	199701	103050	4100
			50	83	201727	241319	98600	1800
		32	100	87	205680	253974	92100	1600
			200	80	176262	190947	112400	1900
			400	85	191591	225796	89400	3000
			50	94	167280	206685	88700	3900
		41	100	90	182084	229433	86300	3200
			200	87	166245	198126	92000	300
			400	89	168366	229950	72700	4100
			50	91	168543	180474	100300	3000
		55	100	87	195111	227507	102300	2900
			200	89	187578	201695	109900	2400
			400	79	214358	241079	94900	3100
			50	93	184461	220184	92100	2900

Problem	Op.	Rates	Op.Size	n	mean	sd	median	min
	TD	14	100	75	215280	234738	123400	5200
			200	76	226470	234050	140600	9700
			400	77	187839	209220	123500	2300
			50	86	214600	223298	133750	3200
		23	100	87	177366	189084	106000	1500
			200	94	215413	252928	97000	800
			400	88	185547	217761	104350	3600
			50	88	198600	245828	87100	2100
		32	100	91	142353	189298	74000	3000
			200	96	187396	215465	91450	400
			400	97	139571	172985	84600	3300
			50	89	192946	238185	80300	2600
	TD	41	100	92	152337	192001	81800	1600
			200	98	115340	141839	66600	1200
			400	100	116725	164787	44650	4800
			50	91	183074	211511	92500	400
		55	100	93	209734	229610	138200	1000
			200	87	211952	224918	115500	3400
			400	90	197939	224316	106550	2200
			50	91	172747	212367	84900	1800
pendulum	GCD	14	other	100	13482	25434	6450	600
		23	other	100	7580	6915	5550	700
		32	other	100	8447	9000	5550	900
		41	other	100	7298	6680	5300	400
		55	other	100	7905	5334	7150	1000
	JD	14	100	100	6502	5887	4700	900
			200	100	9359	12467	5650	500
			400	100	9782	13345	6400	800
			50	100	7502	9275	4600	300
		23	100	100	7179	9004	5500	1000
			200	100	7674	7665	5200	200
			400	100	8107	8159	5900	300

Problem	Op.	Rates	Op.Size	n	mean	sd	median	min
			50	100	7554	7995	4750	400
		32	100	100	7101	6634	4750	400
			200	100	7040	6338	5200	800
			400	100	6698	6381	4350	800
			50	100	6887	6214	5000	900
		41	100	100	6703	6063	5000	800
			200	100	7329	5972	5150	400
			400	100	5921	4672	5050	600
			50	100	6214	7373	4800	400
		55	100	100	7205	6722	5300	400
			200	100	6683	6284	5200	700
			400	100	7077	6715	4900	500
			50	100	5866	5415	4050	400
	TD	14	100	100	7395	6457	5150	1100
			200	100	8355	9499	5750	1000
			400	100	20318	81923	5950	300
			50	100	8574	8857	5700	700
		23	100	100	7103	6510	5000	900
			200	100	6986	6221	4850	100
			400	100	6718	6803	5100	400
			50	100	7137	6236	4800	200
		32	100	100	6116	6858	4400	200
			200	100	6816	6306	4600	400
			400	100	7933	9587	5350	500
			50	100	6716	5852	5250	400
		41	100	100	6956	5738	5100	200
			200	100	7140	6356	5350	400
			400	100	7107	6542	5000	700
			50	100	7038	7521	5150	400
		55	100	100	8459	7879	5750	900
			200	100	5882	5915	4350	200
			400	100	10355	16816	5900	200

Problem	Op.	Rates	Op.Size	n	mean	sd	median	min
			50	100	7513	8943	4700	900
polinomial	GCD	14	other	98	99019	169192	24300	200
		23	other	99	73864	122857	19000	1100
		32	other	97	77951	171638	11500	1100
		41	other	98	84059	157914	19700	500
		55	other	98	73063	112767	25950	800
	JD	14	100	95	70194	142283	16600	1800
			200	91	96502	159656	22000	1300
			400	94	109971	182039	24050	100
			50	96	78851	143225	20500	200
		23	100	93	125342	204288	21900	600
			200	91	89093	163164	17600	1900
			400	93	93044	163854	27400	1300
			50	97	102342	183919	16500	900
		32	100	92	101664	177648	24000	300
			200	89	128078	197466	22300	1300
			400	91	182801	254668	33900	900
			50	95	85295	151331	19300	400
		41	100	97	116162	193904	17400	400
			200	92	107585	195085	21000	1300
			400	83	115401	196940	21500	1000
			50	92	96197	162986	26000	300
		55	100	94	97834	201617	13500	900
			200	93	99240	174813	14900	1100
			400	95	106785	190604	28100	300
			50	93	123237	227992	24500	1100
	TD	14	100	95	86493	166484	16000	200
			200	94	74745	165458	18700	1300
			400	100	90201	162899	25150	1100
			50	100	44115	92876	13800	500
		23	100	96	69020	135604	14550	600
			200	99	76910	157627	17200	200

Problem	Op.	Rates	Op.Size	n	mean	sd	median	min
			400	99	55495	125444	11900	200
			50	97	73994	129626	17100	1300
		32	100	100	61988	124277	14150	100
			200	99	48234	90248	13000	500
			400	99	82492	148358	11800	300
			50	100	88581	160445	27400	1700
		41	100	98	74642	141258	18000	1500
			200	100	46437	78942	12200	500
			400	97	66955	125965	11600	200
			50	97	46851	102973	14300	400
		55	100	99	56018	109088	13800	800
			200	100	82390	145448	15800	500
			400	99	85452	156866	18600	800
			50	98	69040	122151	16400	200
santafetrail	GCD	14	other	98	156062	183619	77650	300
		23	other	96	107096	153006	46100	800
		32	other	98	104526	151325	43150	600
		41	other	99	91581	134514	41100	2100
		55	other	98	108326	149834	45150	900
	JD	14	100	99	115774	162720	49500	1000
			200	93	126617	160887	64400	1300
			400	90	168204	218792	85200	300
			50	100	114803	157577	51300	1900
		23	100	100	118293	183401	41700	2100
			200	97	136079	202116	55700	3100
			400	99	134276	187061	56400	2400
			50	98	100564	128734	45450	3100
		32	100	100	91987	99947	58800	3600
			200	100	94232	150485	39400	1700
			400	97	104519	158176	42000	1200
			50	98	121138	167057	52100	2200
		41	100	99	69427	76134	42200	1600

Problem	Op.	Rates	Op.Size	n	mean	sd	median	min
			200	98	68270	92909	31050	1100
			400	100	86289	114518	40850	1500
			50	100	77954	124846	32150	1100
		55	100	97	118431	172861	54300	1700
			200	97	121013	176845	48800	3100
			400	97	117176	159897	63900	800
			50	98	99891	139062	43750	1300
	TD	14	100	99	95881	115408	52400	2000
			200	98	115971	152992	48600	1400
			400	97	115028	150499	70100	1400
			50	99	107813	138621	52100	2600
		23	100	99	138353	167405	65900	500
			200	99	102665	143457	48700	2200
			400	99	110231	142079	61200	300
			50	99	133382	174059	60300	2700
		32	100	96	129911	173209	61800	1700
			200	96	90411	128585	41000	1000
			400	100	84268	118954	44350	1300
			50	98	116948	163460	57800	900
		41	100	99	89938	123661	45800	600
			200	99	99363	105340	62000	1500
			400	97	110934	164791	55400	1400
			50	99	116661	151654	64200	2600
		55	100	97	135514	189976	50700	2500
			200	100	108436	159779	43900	1800
			400	97	88369	133080	42800	1400
			50	98	127205	175522	63750	1400

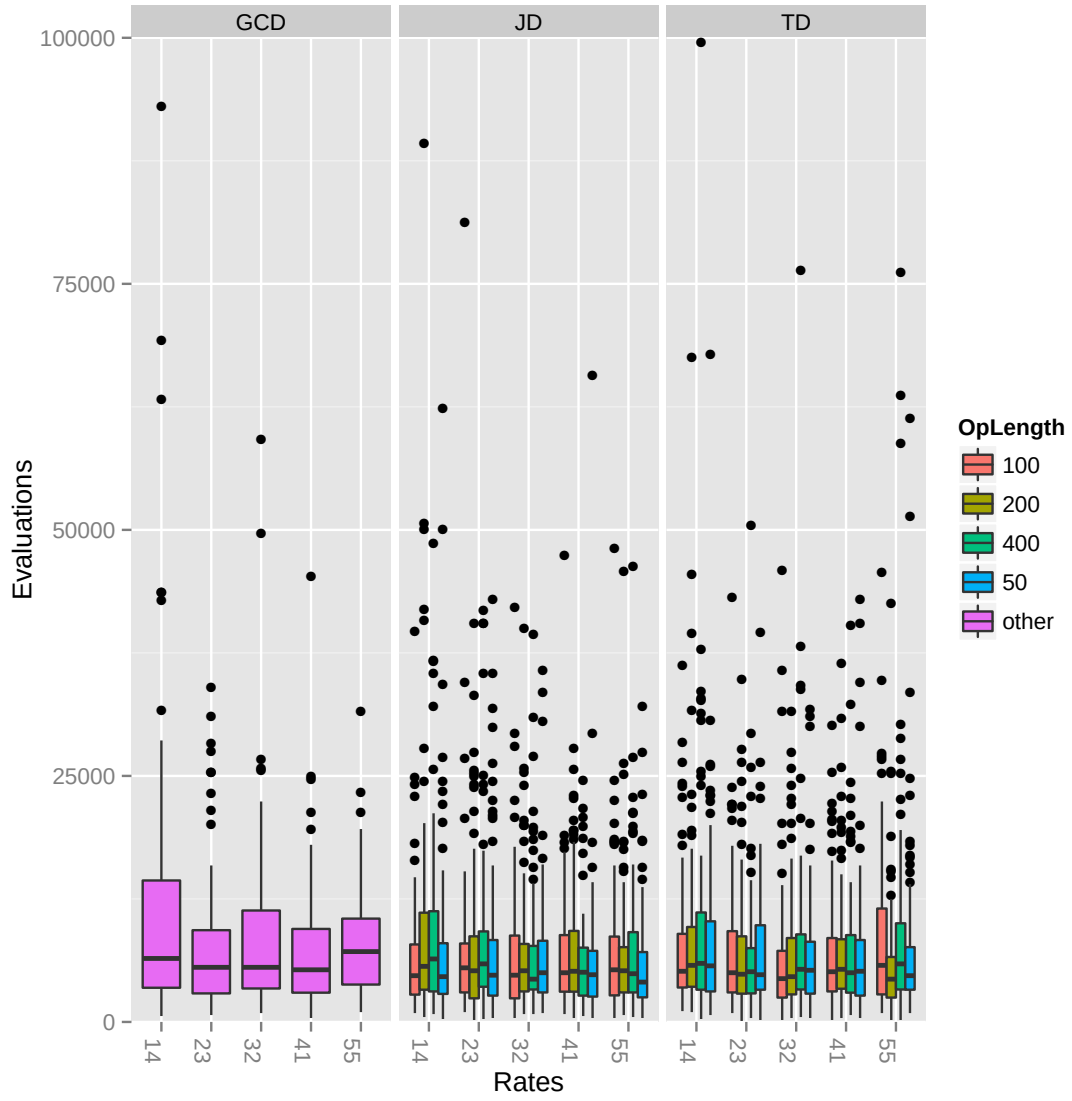


Figure A.6: Efficiency of the asexual operators with the inverted pendulum problem (see Section 6.2). The results are split by operator type and section length (OpLength), where *JD, TD*, and *GCD* correspond respectively to the bitwise *junk* and *tranposon* (coupled with the *delete* operator), and to the *genecopy* and *genedelete* operators. The rates are represented by the first decimal place, that is, *14* stands for *0.1-0.4*, the order consistent with the operators labels.

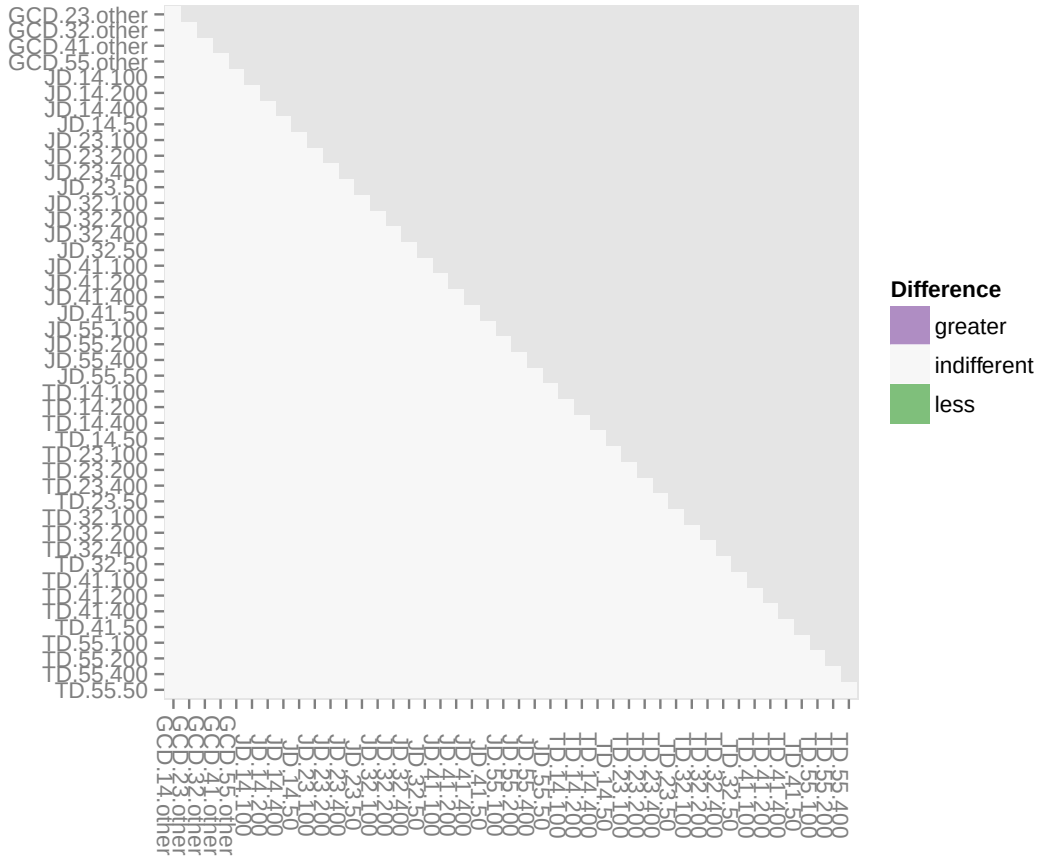


Figure A.7: Pairwise statistical tests results of the efficiency of the asexual operators with the inverted pendulum problem (see Section 6.2). It shows *less* if the number of evaluations of the experiment in the *y* axis is significantly less than the one on the *x* axis, with *greater* otherwise, or *not significant* if the difference is not statistically significant. In the labels *JD, TD*, and *GCD* correspond respectively to the bitwise *junk* and *tranposon* (coupled with the *delete* operator), and to the *genecopy* and *genedelete* operators. The rates are represented by the first decimal place, that is, *14* stands for *0.1-0.4*, the order consistent with the operators labels. The section length is appended at the end.

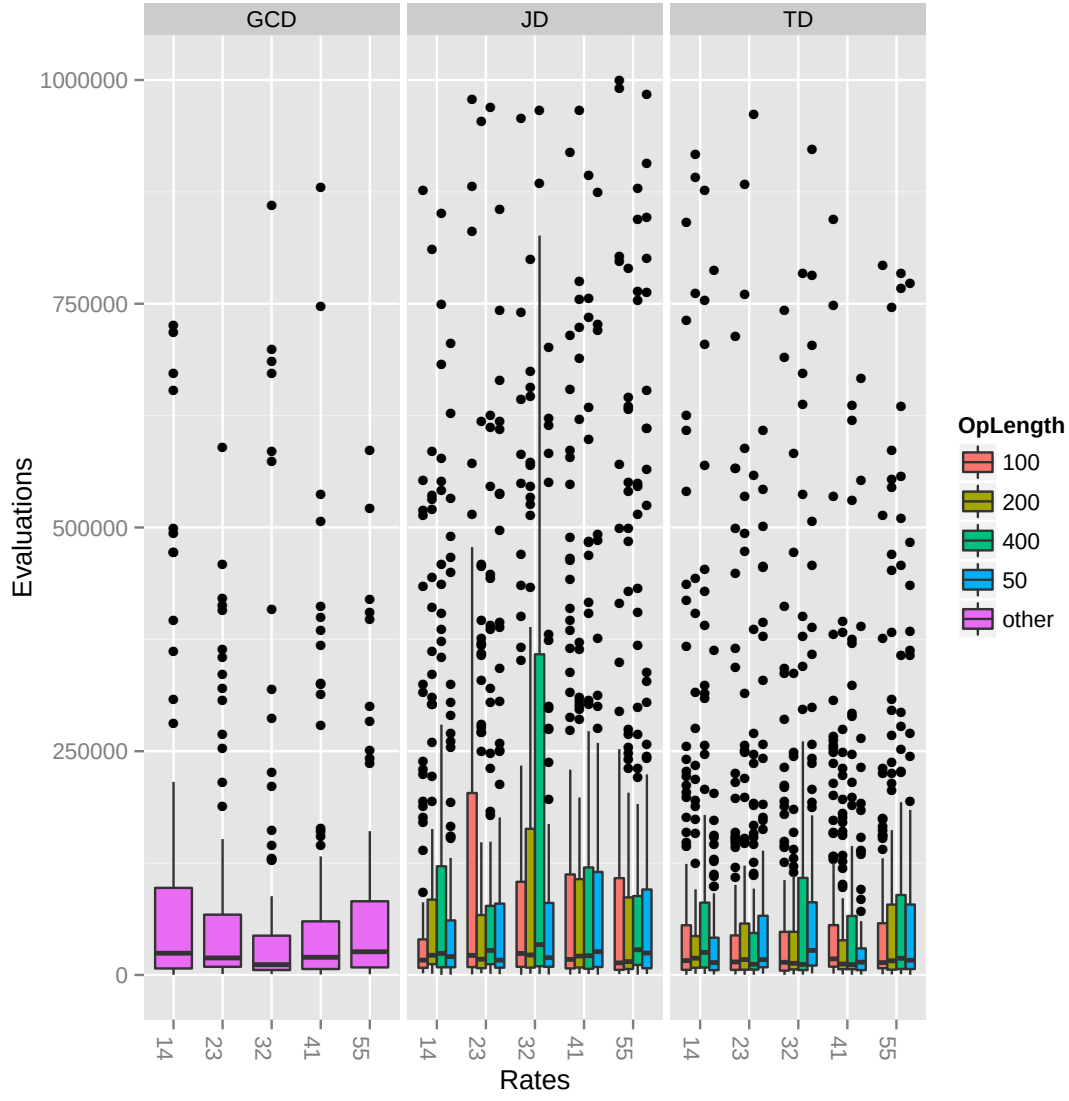


Figure A.8: Efficiency of the asexual operators with the polinomial regression problem (see Section 6.2). The results are split by operator type and section length (OpLength), where *JD, TD*, and *GCD* correspond respectively to the bitwise *junk* and *tranposon* (coupled with the *delete* operator), and to the *genecopy* and *genedelete* operators. The rates are represented by the first decimal place, that is, *14* stands for *0.1-0.4*, the order consistent with the operators labels.

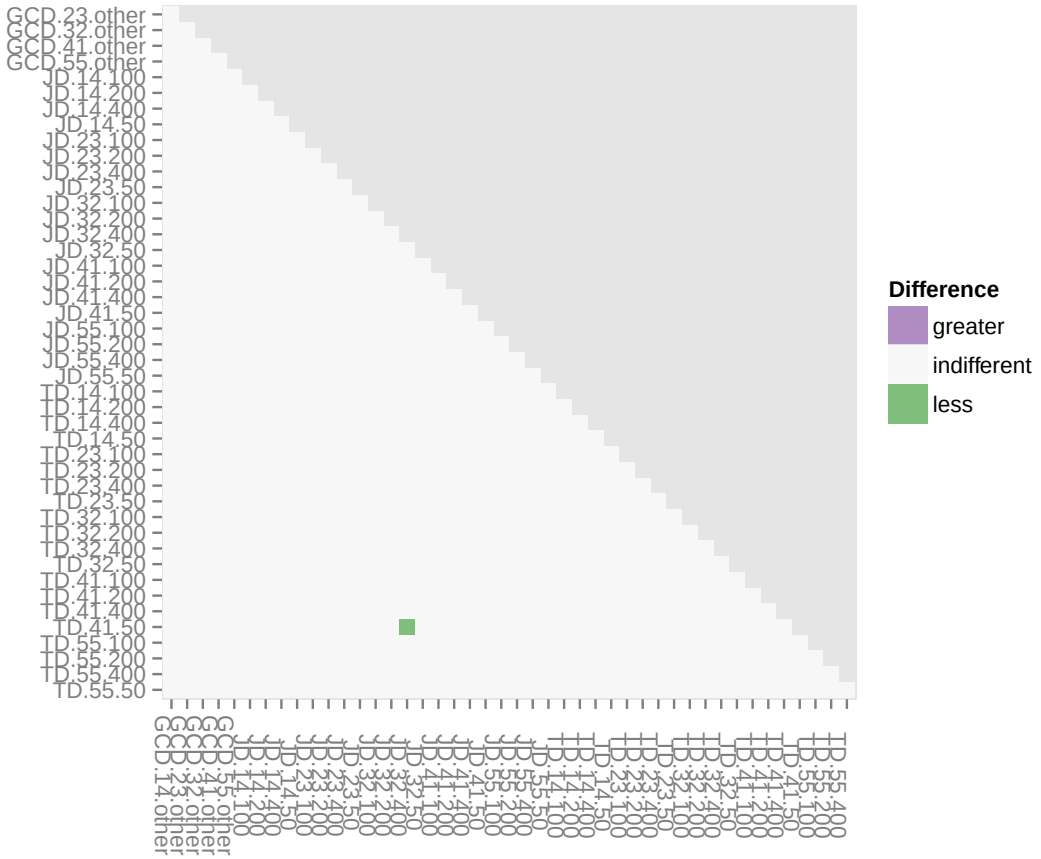


Figure A.9: Pairwise statistical tests results of the efficiency of the asexual operators with the polinomial regression problem (see Section 6.2). It shows *less* if the number of evaluations of the experiment in the *y* axis is significantly less than the one on the *x* axis, with *greater* otherwise, or *not significant* if the difference is not statistically significant. In the labels *JD, TD*, and *GCD* correspond respectively to the bitwise *junk* and *tranposon* (coupled with the *delete* operator), and to the *genecopy* and *genedelete* operators. The rates are represented by the first decimal place, that is, *14* stands for *0.1-0.4*, the order consistent with the operators labels. The section length is appended at the end.

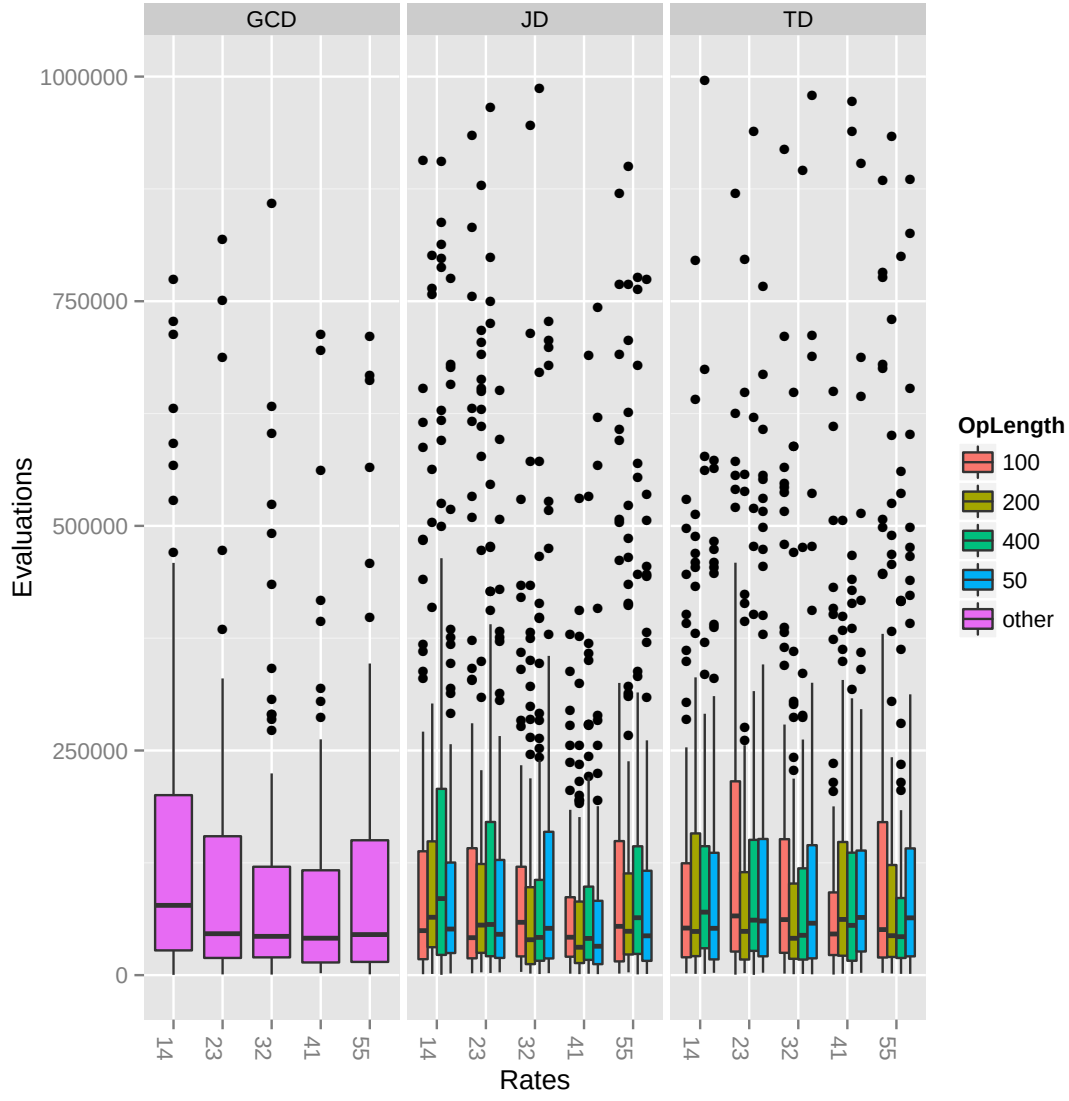


Figure A.10: Efficiency of the asexual operators with the Santa Fe trail problem (see Section 6.2). The results are split by operator type and section length (OpLength), where *JD, TD*, and *GCD* correspond respectively to the bitwise *junk* and *tranposon* (coupled with the *delete* operator), and to the *genecopy* and *genedelete* operators. The rates are represented by the first decimal place, that is, *14* stands for *0.1-0.4*, the order consistent with the operators labels.

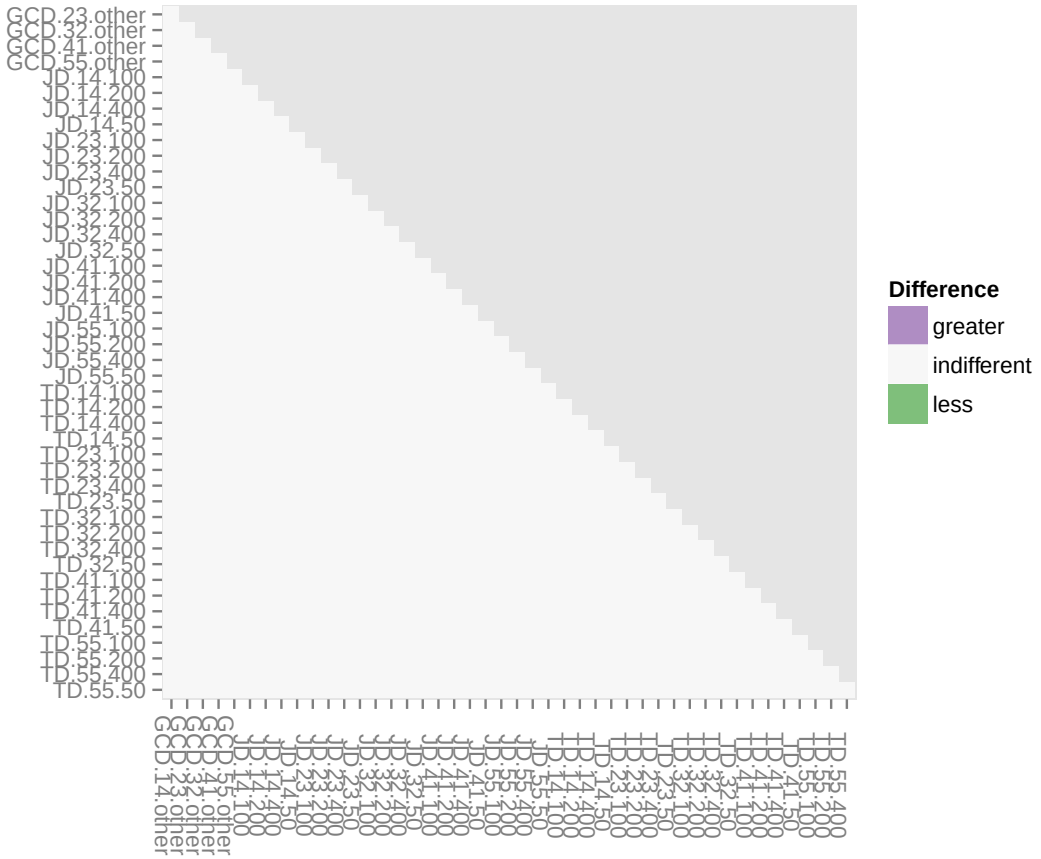


Figure A.11: Pairwise statistical tests results of the efficiency of the asexual operators with the Santa Fe trail problem (see Section 6.2). It shows *less* if the number of evaluations of the experiment in the *y* axis is significantly less than the one on the *x* axis, with *greater* otherwise, or *not significant* if the difference is not statistically significant. In the labels *JD, TD*, and *GCD* correspond respectively to the bitwise *junk* and *tranposon* (coupled with the *delete* operator), and to the *genecopy* and *genedelete* operators. The rates are represented by the first decimal place, that is, *14* stands for *0.1-0.4*, the order consistent with the operators labels. The section length is appended at the end.

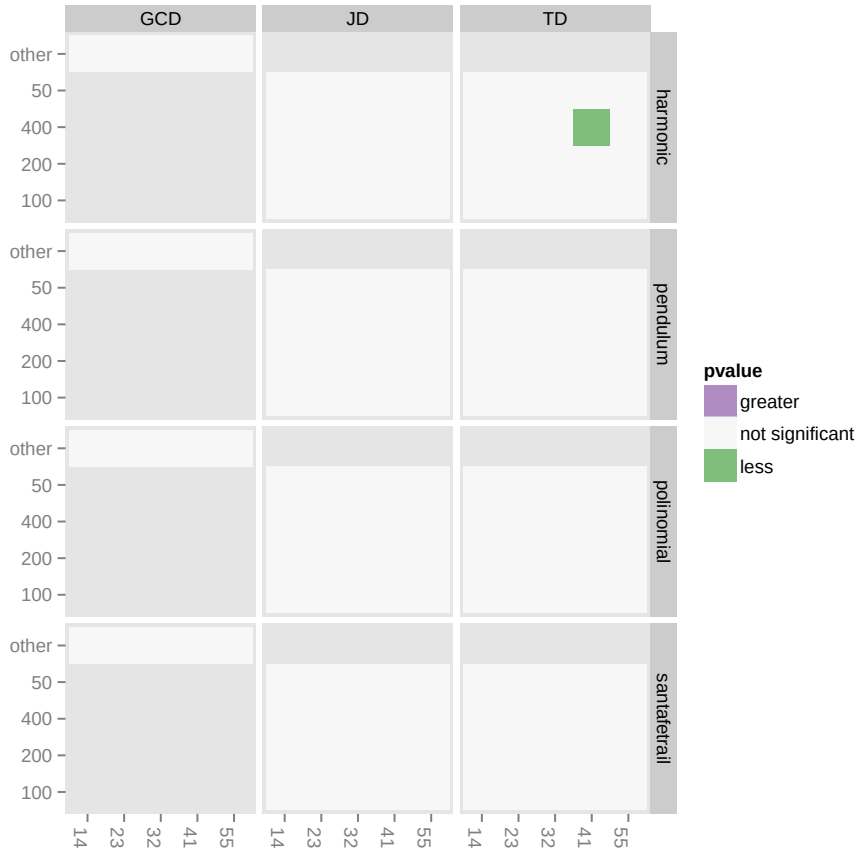


Figure A.12: Comparison of the efficiency results from each asexual operator against the baseline (*rnd7F*), showing *less* if the number of evaluations of the experiment is significantly less than the baseline, with *greater* otherwise, or *not significant* if the difference is not statistically significant. The results are split by problem and operator, with the rates of application on the x axis, and the section length on the y axis. In the labels *JD, TD*, and *GCD* correspond respectively to the bitwise *junk* and *tranposon* (coupled with the *delete* operator), and to the *genecopy* and *genedelete* operators. The rates are represented by the first decimal place, that is, *14* stands for *0.1-0.4*, the order consistent with the operators labels.

Table A.4: Summary of the number of evaluations necessary to find an optimal solution for each crossover operator in every problem. The results are grouped by problem and operator, where *1P*, *2P*, and *uni* correspond respectively to the bitwise *1-point*, *2-point*, and *uniform* crossover operators, with the suffix *gene* applied to the genewise versions.

Problem	Op.	Rate	n	mean	sd	median	min	se
harmonic	1P	0.1	88	162877	194607	96200	400	20745
		0.3	95	176988	208300	95600	1300	21371
		0.5	89	162396	182416	115000	1800	19336
		0.7	89	156020	166938	99100	900	17695
		0.9	94	185513	223280	85100	1800	23030
	1Pgene	0.1	89	180861	204291	98400	400	21655
		0.3	87	144517	183405	75100	1000	19663
		0.5	85	133326	163783	72900	2700	17765
		0.7	86	159377	198950	69900	2100	21453
		0.9	90	171964	196779	101900	2600	20742
	2P	0.1	88	177710	215307	93850	2000	22952
		0.3	90	148421	171781	84400	5200	18107
		0.5	90	134860	189654	57050	1700	19991
		0.7	86	222823	243727	124550	2000	26282
		0.9	82	179444	217565	90750	1900	24026
	2Pgene	0.1	87	167515	193230	124800	800	20716
		0.3	88	144595	183904	78750	2000	19604
		0.5	90	165371	183866	91800	1200	19381
		0.7	84	151011	189109	77850	1600	20633
		0.9	88	214969	240910	123700	400	25681
	uni	0.1	87	157307	178623	81300	3800	19150
		0.3	91	185174	219806	80300	1900	23042
		0.5	92	123902	143122	72500	500	14922
		0.7	88	192489	240348	95800	1900	25621
		0.9	89	143587	179956	76900	1500	19075
	unigene	0.1	49	156012	171783	93000	700	24540
		0.3	31	284684	324955	156500	2500	58364
		0.5	34	264385	273922	158900	6700	46977

Problem	Op.	Rate	n	mean	sd	median	min	se
		0.7	26	246562	300909	84300	6100	59013
		0.9	19	297211	347018	122500	5300	79611
pendulum	1P	0.1	99	7948	6993	5400	1200	703
		0.3	100	8314	9866	5950	400	987
		0.5	100	6876	6063	5500	800	606
		0.7	100	9367	13896	4800	600	1390
		0.9	100	11288	26088	5300	600	2609
	1Pgene	0.1	100	6540	5000	4500	700	500
		0.3	100	7389	7923	5150	300	792
		0.5	100	8409	12968	5200	600	1297
		0.7	99	12481	23317	5400	500	2343
		0.9	100	13174	22546	6250	500	2255
	2P	0.1	100	7065	8520	4800	600	852
		0.3	100	8241	10947	5800	500	1095
		0.5	100	9279	19643	4650	900	1964
		0.7	100	8640	12992	5100	400	1299
		0.9	100	11174	17054	6150	300	1705
	2Pgene	0.1	98	10604	36640	4400	600	3701
		0.3	97	7056	8508	4000	900	864
		0.5	100	7989	7746	5450	600	775
		0.7	97	9107	10840	5300	300	1101
		0.9	93	8989	10469	5800	600	1086
	uni	0.1	100	6816	6077	4850	600	608
		0.3	100	7098	6306	5050	800	631
		0.5	100	7583	7197	5500	600	720
		0.7	100	7640	7966	5150	900	797
		0.9	100	10690	16933	6100	700	1693
	unigene	0.1	100	7498	9045	4500	200	904
		0.3	96	14529	23832	6650	300	2432
		0.5	90	21811	101585	6350	800	10708
		0.7	87	31545	84017	5400	200	9008

Problem	Op.	Rate	n	mean	sd	median	min	se
		0.9	80	99628	199381	10450	300	22291
polinomial	1P	0.1	99	67572	127894	18300	100	12854
		0.3	97	87012	166667	13700	100	16922
		0.5	95	100741	198086	20300	100	20323
		0.7	99	94557	170920	20200	100	17178
		0.9	97	85654	162044	15000	100	16453
	1Pgene	0.1	98	72781	170684	18150	100	17242
		0.3	99	56863	105628	12500	100	10616
		0.5	92	58235	120336	12900	100	12546
		0.7	100	107489	213060	13200	100	21306
		0.9	99	87064	166356	13900	100	16719
	2P	0.1	100	93460	174873	19000	100	17487
		0.3	100	73707	160260	13750	100	16026
		0.5	98	97897	157128	22300	100	15872
		0.7	99	79988	156752	15800	100	15754
		0.9	99	78081	150903	15500	100	15166
	2Pgene	0.1	99	88453	168801	14900	100	16965
		0.3	99	55278	100419	13000	100	10092
		0.5	97	62373	125545	16200	100	12747
		0.7	100	103494	188872	20200	100	18887
		0.9	99	126870	211770	16100	100	21284
	uni	0.1	99	70137	134862	19400	100	13554
		0.3	96	74040	131723	14850	100	13444
		0.5	98	67812	122872	14150	100	12412
		0.7	97	100089	196968	20400	100	19999
		0.9	99	104264	169065	18800	100	16992
	unigene	0.1	79	96762	172410	18400	100	19398
		0.3	79	102511	192743	29900	100	21685
		0.5	61	98851	146292	30800	100	18731
		0.7	61	131480	179045	62700	100	22924
		0.9	59	123317	176451	42700	100	22972
santafetrail	1P	0.1	96	103008	122106	55900	2300	12462

Problem	Op.	Rate	n	mean	sd	median	min	se
		0.3	97	158592	215375	74600	600	21868
		0.5	98	97270	124600	50600	300	12586
		0.7	96	122852	172517	52800	1600	17607
		0.9	95	116949	147755	60100	3800	15159
	1Pgene	0.1	99	128879	167825	67100	600	16867
		0.3	96	119890	182818	52350	2500	18659
		0.5	99	113890	167049	49400	1900	16789
		0.7	98	140607	177010	67900	1600	17881
		0.9	96	132772	187143	58350	1300	19100
	2P	0.1	97	125575	139335	69500	900	14147
		0.3	97	121991	158145	54400	2600	16057
		0.5	94	131297	189348	49650	2900	19530
		0.7	100	102584	133906	51600	300	13391
		0.9	98	134617	162658	72400	1400	16431
	2Pgene	0.1	98	128122	173681	49000	2400	17544
		0.3	98	141537	162903	83850	1600	16456
		0.5	97	120663	161420	61400	1900	16390
		0.7	98	136723	174273	70300	200	17604
		0.9	100	136829	176997	65300	2100	17700
	uni	0.1	90	151003	207666	67250	1100	21890
		0.3	98	114629	143789	67200	1400	14525
		0.5	97	121549	154496	67300	1300	15687
		0.7	99	142509	155257	91300	1600	15604
		0.9	100	100414	154179	37350	700	15418
	unigene	0.1	82	133704	175022	59950	1100	19328
		0.3	62	173711	198962	124050	3400	25268
		0.5	63	180851	224671	95000	800	28306
		0.7	53	231719	262548	124100	1200	36064
		0.9	59	234481	246403	161600	5000	32079

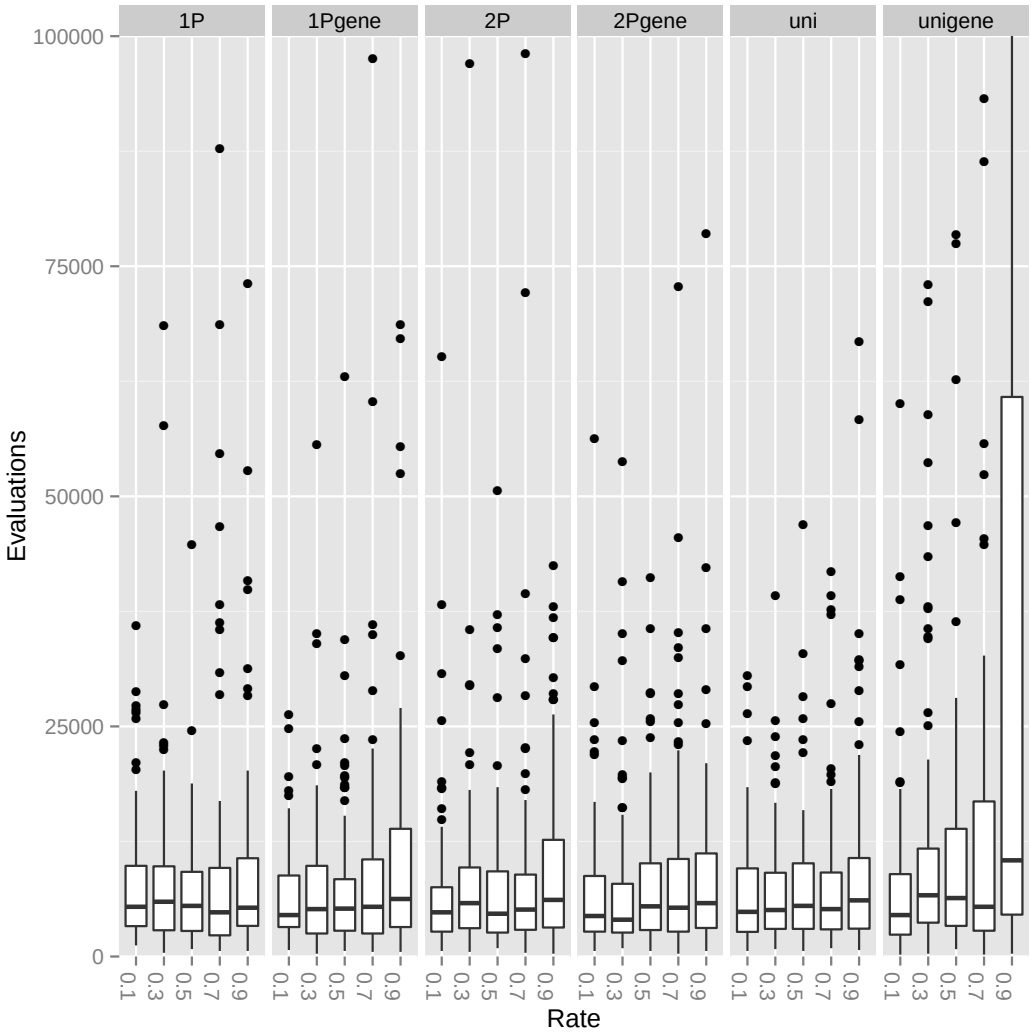


Figure A.13: Distribution of the number of evaluations necessary to find an optimal solution for the inverted pendulum problem, using crossover operators. The results are split by operator type, where *1P*, *2P*, and *uni* correspond respectively to the bitwise *1-point*, *2-point*, and *uniform* crossover operators, with the suffix *gene* applied to the genewise versions.

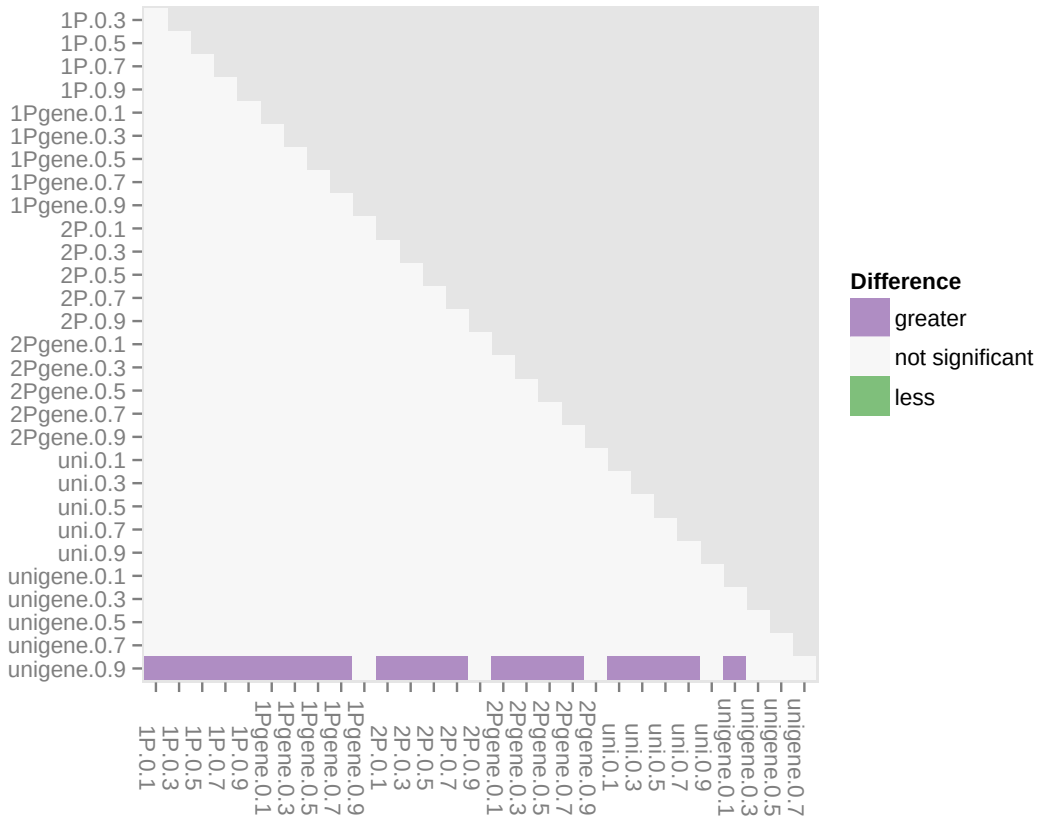


Figure A.14: Pairwise statistical tests results of the number of evaluations necessary to find an optimal solution for the inverted pendulum problem, using crossover operators. It shows *less* if the number of evaluations of the experiment in the *y* axis is significantly less than the one on the *x* axis, with *greater* otherwise, or *not significant* if the difference is not statistically significant. The results are split by operator and rate of application, where *1P*, *2P*, and *uni* correspond respectively to the bitwise *1-point*, *2-point*, and *uniform* crossover operators, with the suffix *gene* applied to the genewise versions.

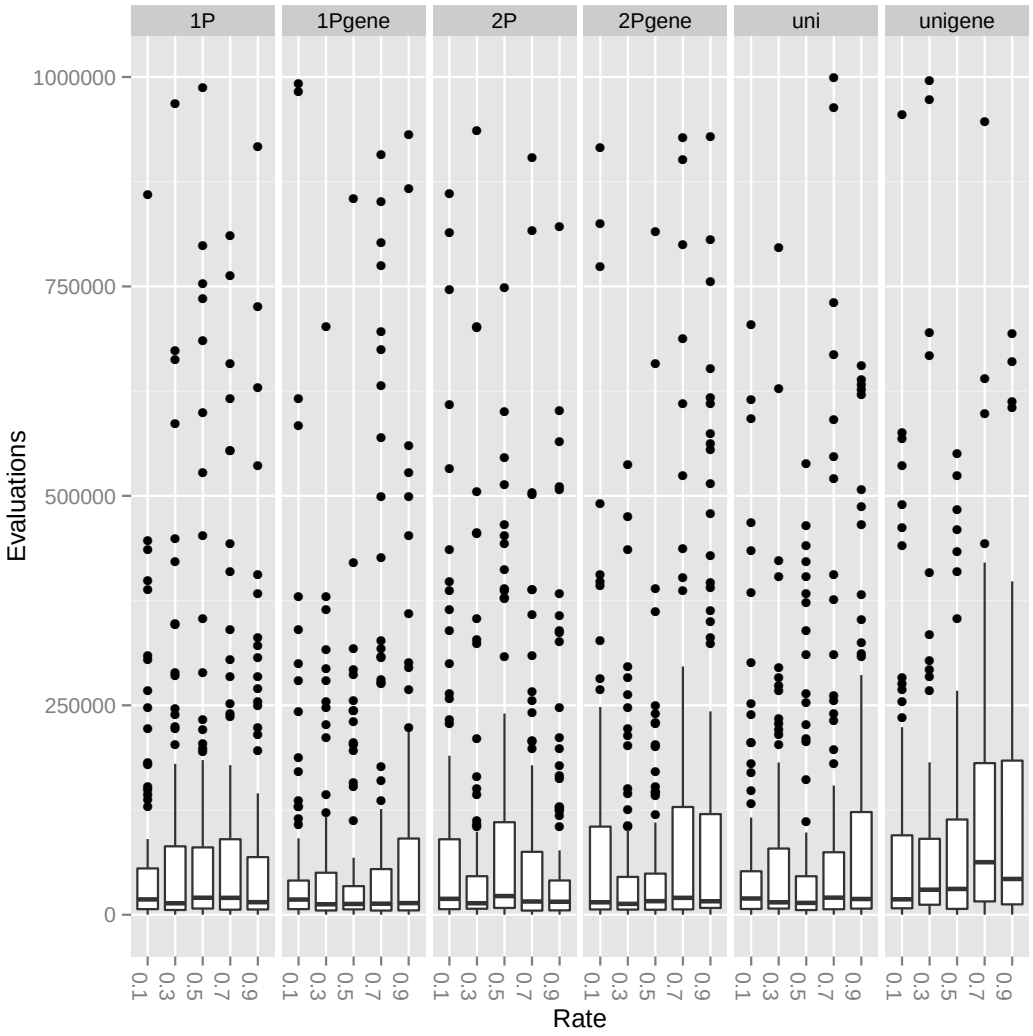


Figure A.15: Distribution of the number of evaluations necessary to find an optimal solution for the polinomial regression problem, u sing crossover operators. The results are split by operator, where *1P*, *2P*, and *uni* correspond respectively to the bitwise *1-point*, *2-point*, and *uniform* crossover operators, with the suffix *gene* applied to the genewise versions.

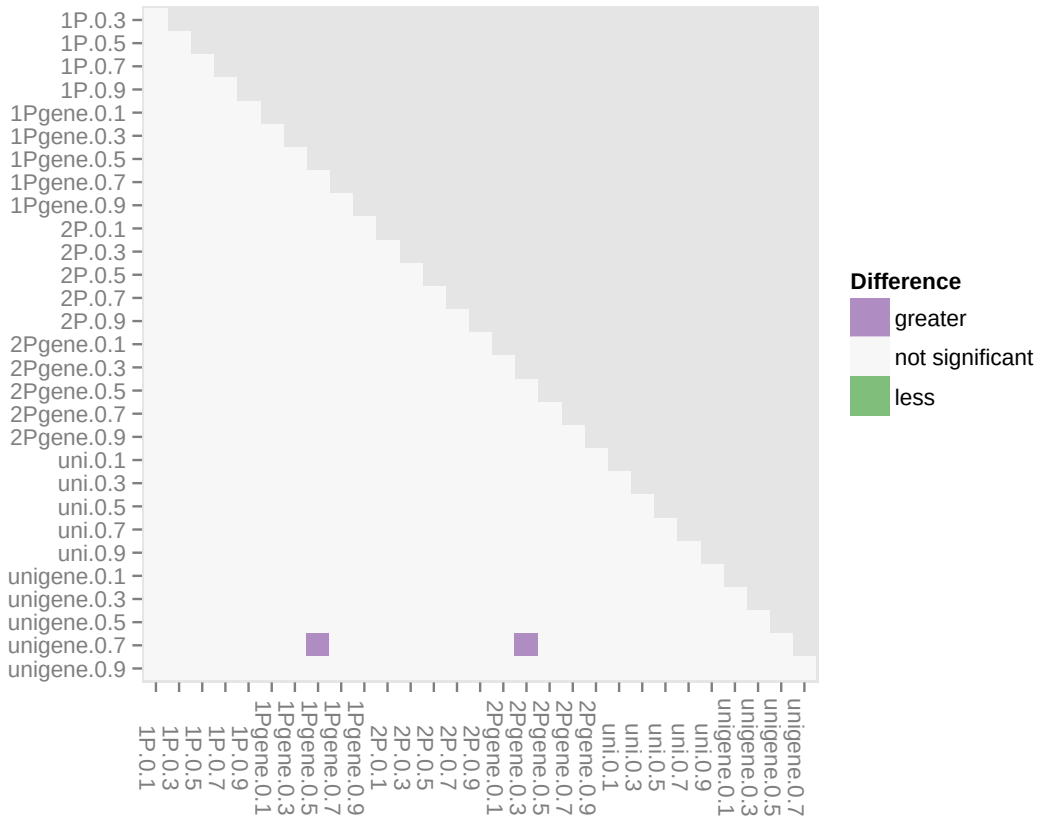


Figure A.16: Pairwise statistical tests results of the number of evaluations necessary to find an optimal solution for the polinomial regression problem, using crossover operators. It shows *less* if the number of evaluations of the experiment in the *y* axis is significantly less than the one on the *x* axis, with *greater* otherwise, or *not significant* if the difference is not statistically significant. The results are split by operator and rate of application, where *1P*, *2P*, and *uni* correspond respectively to the bitwise *1-point*, *2-point*, and *uniform* crossover operators, with the suffix *gene* applied to the genewise versions.

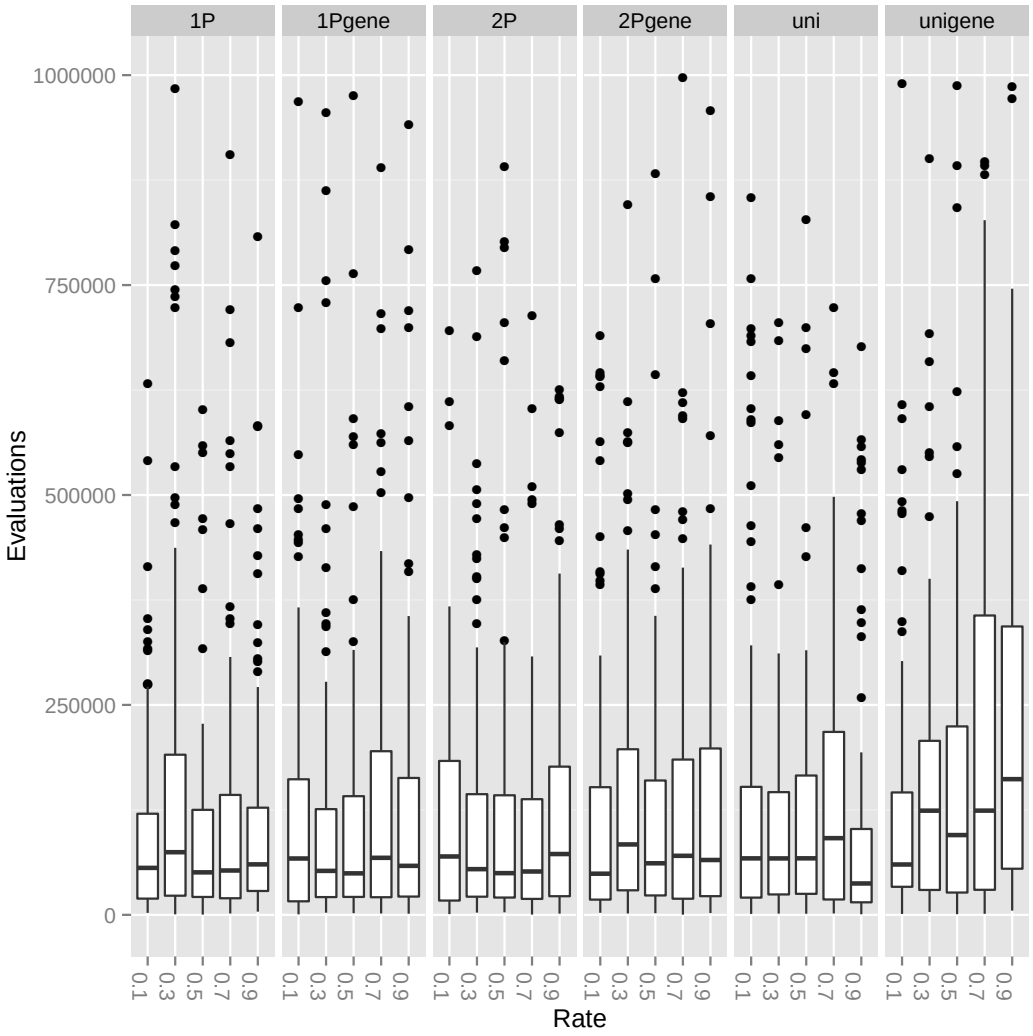


Figure A.17: Distribution of the number of evaluations necessary to find an optimal solution for the Santa Fe trail problem, using crossover operators. The results are split by operator, where *1P*, *2P*, and *uni* correspond respectively to the bitwise *1-point*, *2-point*, and *uniform* crossover operators, with the suffix *gene* applied to the genewise versions.

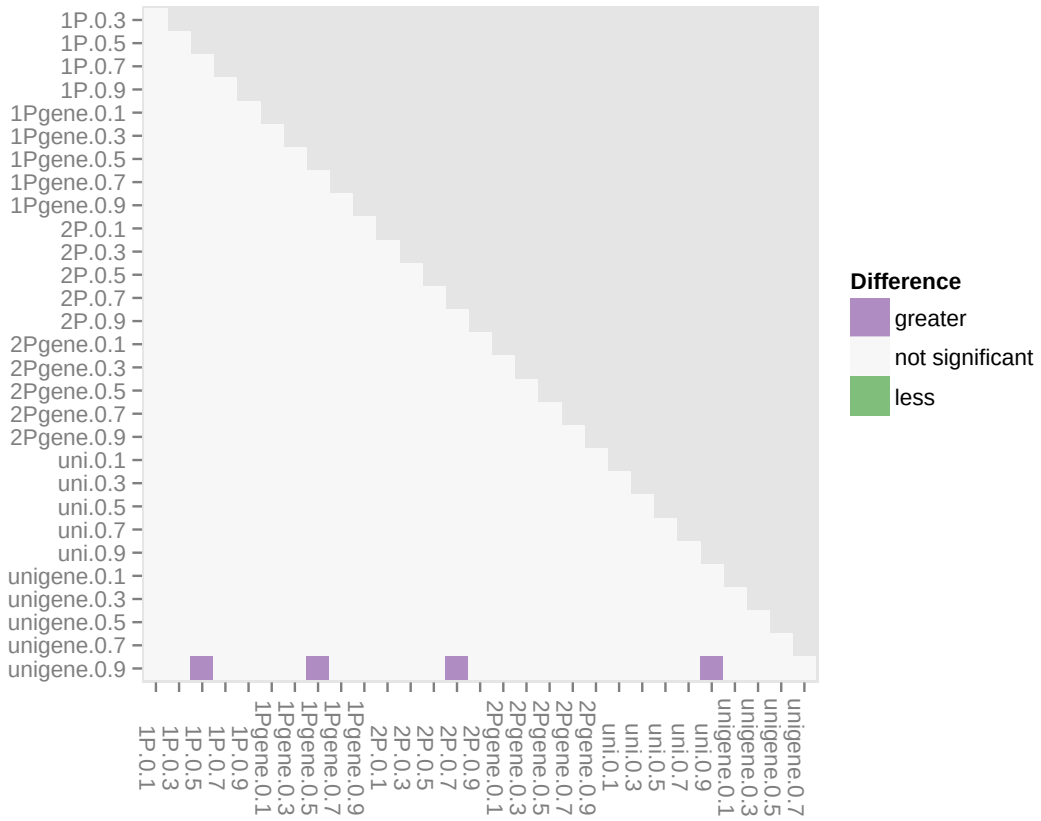


Figure A.18: Pairwise statistical tests results of the number of evaluations necessary to find an optimal solution for the Santa Fe trail problem, using crossover operators. It shows *less* if the number of evaluations of the experiment in the *y* axis is significantly less than the one on the *x* axis, with *greater* otherwise, or *not significant* if the difference is not statistically significant. The results are split by operator and rate of application, where *1P*, *2P*, and *uni* correspond respectively to the bitwise *1-point*, *2-point*, and *uniform* crossover operators, with the suffix *gene* applied to the genewise versions.