An Exploration of Tree-Adjoining Grammars for Grammatical Evolution

Eoin Murphy

B.Sc. University College Dublin

Thesis submitted to University College Dublin

for the degree of Ph.D.

at the School of Computer Science and Informatics

Research Supervisors:

Prof. Michael O'Neill

Prof. Anthony Brabazon

External Examiner:

Prof. Ernesto Costa

December 6, 2013

Abstract

Grammars are an important tool for Evolutionary Computation (EC). Grammars offer a flexible means of search space restriction, and provide a mechanism for imposing language and search biases. This thesis explores the use of a particular grammar type, Tree-Adjoining Grammars (TAGs) for use with Grammatical Evolution (GE), a grammar-based EC algorithm. To date, much of the work on GE has used Context-Free Grammars (CFGs). TAGs have been shown to be more powerful than CFGs and can generate some context-sensitive languages. TAGs also exhibit interesting properties, such as, each intermediate stage of TAG derivation being a fully structured feasible sentence from the language.

The focus of this thesis is to explore the use of TAGs for representation in GE. A definition of TAGs is given and a comprehensive survey of TAGs in EC is presented. Following this, a novel representation and mapping process is developed which combines the linear chromosome used by GE with TAGs. This extension of GE, called Tree-Adjoining Grammatical Evolution (TAGE), is compared with canonical GE on a number of benchmark problems. TAGE demonstrates a performance benefit over GE. Further study of the two representations is presented, which identifies core representational differences, such as, invalid individuals and neutral crossover operations, both of which do not occur in TAGE. These differences are shown to account for some of improved performance of TAGE.

Subsequent to this, a novel method of rendering search landscapes is presented. Single mutation event landscapes are generated for GE and TAGE for a number of common grammars. It is shown that TAGE search spaces are much more densely connected than those of GE, affording TAGE greater opportunities to move about the search space.

Further representational differences in the form of preferential language biases are discovered when developing methods of generating similar initial populations for both TAGE and GE. Two main biases are identified, adjunction constraints and grammar transformation biases. These biases affect the distributions of tree structures generated by TAGE.

Methods of mitigating these biases are presented and it is shown that these biases can provide problem dependent performance benefits.

The developmental nature of the feasibility property of TAGs is exploited by integrating an online artificial gene regulatory network (GRN) model with TAGE in the form of Developmental TAGE (DTAGE). DTAGE is shown to improve the usability of this GRN model by facilitating it with the use of the TAGE mapping process. DTAGE is demonstrated to be capable of evolving GRNs whose output, when provided with feedback in the form of state information from dynamic problem environments, maps to phenotypes that can survive in those environments.

In summary, this thesis explores the utility and capability of TAGs for representation in GE. Differences in representation between TAGs and CFGs for use with GE are identified and studied in terms of performance. TAGs are then exploited in the development of a novel evolutionary developmental system combining TAGs, GE and a GRN model.

Acknowledgements

I would first like to thank to my supervisors, Professor Michael O'Neill and Professor Anthony Brabazon for their guidance and support throughout, without which the work presented within would not have been possible; Michael, for his boundless enthusiasm, guidance and insight; Tony for his keen eye for detail, for driving me by always asking the "hard questions", and inspiring me through his interest in nature inspired algorithms.

I want to thank all of the members, past and present, of the Natural Computing Research and Applications group for creating a working environment that has allowed me to learn from each of them (and impart some of my own knowledge too), as well as providing encouragement in both my academic and personal interests. They are a pleasure to work with and I would like to thank them all for their friendship.

In particular, I would like to express my thanks to Dr. Jonathan Byrne, Dr. Erik Hemberg and Dr. Miguel Nicolau for endless discussion and philosophising of both relevant and not-so-relevant topics and theories. Special thanks for all of their rubber ducking of ideas, reviewing of research, and for all of the guidance and encouragement they have provided, particularly with regards to completing this thesis.

Additional thanks to Dr. David Fagan with whom I have shared eight years of education with. I think it is safe to say that without Dave, I would not have made it this far, or if I did, I would have used my bike far more often. I'd like to thank him for all his lifts, tea breaks, pool breaks and table tennis games.

I would also like to thank my parents, for their support and encouragement throughout my education, and for keeping a roof above my head during my Ph.D. As well as thanks to Emma, who kept me sane from afar, and when schedules allowed, from not-so afar.

Finally, I would like to express my thanks to the Science Foundation Ireland (SFI). This research is based upon works supported by the SFI under Grant No. 08/IN.1/I1868.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Publications Arising

1. Eoin Murphy, Michael O'Neill, Edgar Galvan-Lopez, and Anthony Brabazon. Tree-adjunct grammatical evolution. In *2010 IEEE World Congress on Computational Intelligence*, pages 4449–4456, Barcelona, Spain, 18-23 July 2010. IEEE Computational Intelligence Society, IEEE Press. doi: 10.1109/CEC.2010.5586497

2. Edgar Galvan-Lopez, David Fagan, Eoin Murphy, John Mark Swafford, Alexandros Agapitos, Michael O'Neill, and Anthony Brabazon. Comparing the performance of the evolvable PiGrammatical evolution genotype-phenotype map to grammatical evolution in the dynamic Ms. Pac-Man environment. In *2010 IEEE World Congress on Computational Intelligence*, pages 1587–1594, Barcelona, Spain, 18-23 July 2010. IEEE Computational Intelligence Society, IEEE Press. doi: 10.1109/CEC.2010.5586508

3. Petr Pospichal, Eoin Murphy, Michael O'Neill, Josef Schwarz, and Jiri Jaros. Acceleration of grammatical evolution using graphics processing units: computational intelligence on consumer games and graphics hardware. In Simon Harding, W. B. Langdon, Man Leung Wong, Garnett Wilson, and Tony Lewis, editors, *GECCO 2011 Computational intelligence on consumer games and graphics hardware (CIGPU)*, pages 431–438, Dublin, Ireland, 12-16 July 2011. ACM. doi: 10.1145/2001858.2002030

4. Eoin Murphy. Examining grammars and grammatical evolution in dynamic environments. In Miguel Nicolau, editor, *GECCO 2011 Graduate students workshop*, pages 779–782, Dublin, Ireland, 12-16 July 2011. ACM. doi: 10.1145/2001858.2002090

5. Eoin Murphy, Michael O'Neill, and Anthony Brabazon. A comparison of GE and TAGE in dynamic environments. In Natalio Krasnogor, Pier Luca Lanzi, Andries Engelbrecht, David Pelta, Carlos Gershenson, Giovanni Squillero, Alex Freitas, Marylyn Ritchie, Mike Preuss, Christian Gagne, Yew Soon Ong, Guenther Raidl, Marcus Gallager, Jose Lozano, Carlos Coello-Coello, Dario Landa Silva, Nikolaus Hansen, Silja Meyer-Nieberg, Jim Smith, Gus Eiben, Ester Bernado-Mansilla, Will Browne, Lee Spector, Tina Yu, Jeff Clune, Greg Hornby, Man-Leung Wong, Pierre Collet, Steve Gustafson, Jean-Paul Watson, Moshe Sipper, Simon Poulding, Gabriela Ochoa, Marc Schoenauer, Carsten Witt, and Anne Auger, editors, *GECCO '11: Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1387–1394, Dublin, Ireland, 12-16 July 2011. ACM. doi: 10.1145/2001576.2001763

6. Quang Uy Nguyen, Eoin Murphy, Michael O'Neill, and Xuan Hoai Nguyen. Semantic-based subtree crossover applied to dynamic problems. In Tu Bao Ho, R. I. McKay, Xuan Hoai Nguyen, and The Duy Bui, editors, *The Third International Conference on Knowledge and Systems Engineering, KSE'2011*, pages 78–84, Hanoi University, 14–16 October 2011. IEEE. doi: 10.1109/KSE.2011.20

7. Eoin Murphy, Michael O'Neill, and Anthony Brabazon. Examining mutation landscapes in grammar based genetic programming. In Sara Silva, James A. Foster, Miguel Nicolau, Mario Giacobini, and Penousal Machado, editors, *Proceedings of the 14th European Conference on Genetic Programming, EuroGP 2011*, volume 6621 of *LNCS*, pages 130–141, Turin, Italy, 27-29 April 2011. Springer Verlag. doi: 10.1007/978-3-642-20407-4_12. Best paper

8. Eoin Murphy, Erik Hemberg, Miguel Nicolau, Michael O'Neill, and Anthony Brabazon. Grammar bias and initialisation in grammar based genetic programming. In Alberto Moraglio, Sara Silva, Krzysztof Krawiec, Penousal Machado, and Carlos Cotta, edi-

tors, *Proceedings of the 15th European Conference on Genetic Programming, EuroGP 2012*, volume 7244 of *LNCS*, pages 85–96, Malaga, Spain, 11-13 April 2012. Springer Verlag. doi: 10.1007/978-3-642-29139-5_8

9. Eoin Murphy, Miguel Nicolau, Erik Hemberg, Michael O'Neill, and Anthony Brabazon. Differential gene expression with tree-adjunct grammars. In Carlos A. Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone, editors, *Parallel Problem Solving from Nature, PPSN XII (part 1)*, volume 7491 of *Lecture Notes in Computer Science*, pages 377–386, Taormina, Italy, September 1-5 2012. Springer. doi: 10.1007/978-3-642-32937-1_38

Chapter 1

Introduction

This thesis is based in the field of Evolutionary Computation (EC), a sub-field of Artificial Intelligence. EC is a population-based stochastic search method for automatic problem solving. The driving inspiration for EC is what Spencer coined *"survival of the fittest"*, one of the basic principles of Darwin's theory of evolution by natural selection. It is evident that nature has the ability to find diverse solutions to many problems and, as such, it is natural that the field of Artificial Intelligence should look upon these processes for inspiration. Processes which have been seeking perfection in ever changing environments on this planet for millions, if not billions, of years.

The chosen representation, how solutions are coded, is very important for any problem solving method. This coding must be sufficient to represent a wide range of possible solutions, in addition to ensuring that the search space is navigable. Moreover, in the case of natural selection, the representation must be evolvable, i.e., variation of individuals in the population should result in a net improvement in the population's ability to survive in the changing environment. Both of these properties are seen in nature, in the diversity and continued existence of life.

It is with representation that the focus of this thesis lies. Grammars, a popular and powerful formalism in Computer Science, outline a set of rules and symbols which define a set of structured sentences, i.e., a language. In the context of EC, when used as part of the representation, grammars afford a flexible means of defining and restricting the search space, in addition to providing a trivial method of embedding domain specific information into the solutions. This flexibility allows grammar-based EC algorithms to be applied to a vast range of problem classes. The investigations of this thesis explore the application of a specific type of grammar, tree-adjoining grammars, which have been shown to be more powerful than the more commonly used context-free grammars, for use with a grammar-based EC algorithm, grammatical evolution.

The rest of this chapter proceeds as follows. Section 1.1 introduces EC - the field in which this thesis is set. Section 1.1.1 briefly discuss grammars and how they are beneficial for use as representation in EC. Different variations of grammar-based EC are then described, in particular, grammatical evolution, the form of grammar-based EC utilised by this thesis, is presented. The section goes on to introduce tree-adjoining grammars, the grammar formalism explored throughout this work. Section 1.2 outlines the aims of the thesis, the research questions proposed, and the objectives that the thesis addresses. Section 1.3 gives an overview of the main contributions of the work presented throughout, with the limitations of that work provided in Section 1.4. Finally, Section 1.5 provides a summary of each of the following chapters.

1.1 Evolutionary Computation

One of the core concepts of natural selection is that individuals who are more suited to the [problem] environment in which they find themselves are more likely to survive, and, as a result, these *fit* individuals reproduce more often, passing on their different advantages to their offspring. As such, a parallel trial and error approach is taken in EC, with a population of potential solutions dispersed throughout the search space. The fittest of these solutions are allowed to survive and reproduce. Reproduction is the mechanism by

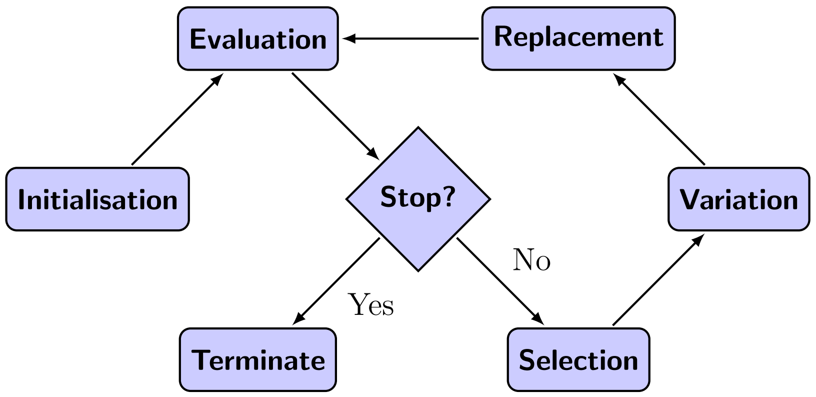


Figure 1.1: Execution flow of a typical EC algorithm.

which new potential solutions are introduced into the population and to further search the space of solutions. Reproductive processes can include: the direct copying of a solution, the slight modification of a solution, as well as the recombination of multiple solutions to name a few. These operations are often referred to as variation operations. Figure 1.1 portrays the typical execution flow of an EC algorithm.

Many different variations of EC algorithms exist. These include, Evolutionary Programming [53] (EP), Evolutionary Strategies [12] (ES), Genetic Algorithms [59] (GA), Genetic Programming [113, 8, 189] (GP) and Differential Evolution [208] (DE), to name a few. Many of these variations have also spawned their own derivative approaches. Different approaches often vary in their method of representing potential solutions, as well as the way in which search is performed using that representation. As such, the classes of problems which each method can be applied to also vary. One of the more popular - and successful [112, 169] - forms of EC algorithms, and that with which this thesis is concerned, is GP, an EC algorithm for automatic programming. Originally popularised by Koza [113], tree-based GP uses a lisp expression-tree, or program-tree, representation for the individuals in its population. While this approach has been very successful, the lack of syntactic and type information built into the basic program-tree representation make GP difficult to apply to certain classes of problems. The focus of this thesis, however, is primarily on a form of grammar-based GP (GBGP) and exploration of the grammars used with this system. Grammars allow the embedding of syntactic and type information within the representation, as well as offering a flexible means of defining, and restricting, the search space. Grammars also make it trivial to embed domain specific information into the representation.

1.1.1 Grammar-Based Genetic Programming

A grammar is a mechanism for the creation of the set of strings of a particular language [69]. Grammars are an important formalism in Computer Science and are widely used to represent restrictions on general domains, as well as to define legal expressions and impose type restrictions. As such, grammars offer many advantages when used as part of a representation for GP [134]. The most prominent advantage is that grammars offer a flexible means of search space restriction, providing a means of keeping the cost of search to the minimum required to find a solution. Moreover, grammars provide a mechanism for controlling different types of bias, language biases, which affect the set of generated sentences in the search space; as well as search biases, which are related to the connectivity of the search space. One of the more commonly used types of grammar are Chomsky grammars, in particular, context-free grammars (CFG).

While many different approaches to GBGP have been developed, two main forms exist. The first is a tree-based representation utilising CFGs, originally proposed by Whigham [215] in a system called CFG-GP. Unlike, tree-based GP, this system separates the *genotype*, the object upon which variation operations are applied, and the *phenotype*, the evaluated program. In the case of CFG-GP, the genotypes are derivation trees, i.e., structured sentences derived from the grammar (see Figure 1.2), and the phenotypes, the sentences, are extracted from the frontiers of those trees. Geyer-Schulz [57] and Wong and Leung [219] also proposed similar systems around the same time, with Wong and Leung using a

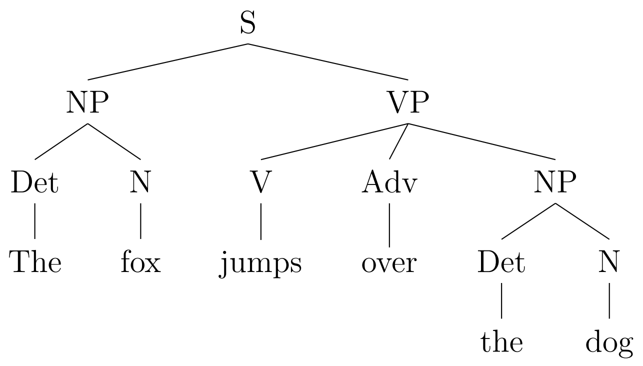


Figure 1.2: A sample derivation tree for the sentence *"The fox jumps over the dog"*.

different, slightly more powerful, grammar type (definite-clause grammars). The second type of GBGP is a linear variation. In this form, the derivation tree has been replaced by a linear genome of bits or integers as the genotype, allowing much theory from the fields of ESs and GAs to be carried over into GP. Early work by Keller and Banzhaf [105] used a grammar to repair sentences generated from a binary genotype. Another linear GBGP system, the system used throughout this thesis, is Grammatical Evolution [163, 172, 173, 42] (GE). McKay et al. [134], and more recently Hemberg [70], provide detailed surveys on the use of grammars for GP.

Grammatical Evolution

Proposed by O'Neill and Ryan [172], GE is one of the most widely used GP systems today [134]. It is a variable length, linear form of GBGP that makes use of a linear genome. The genome is interpreted by a CFG and used to construct a derivation tree. The derivation tree is an intermediate state of representation, as a phenotype is then extracted from the leaf nodes of this tree. GE's novelty comes from its use of a variable length genotype, the redundant genotype-phenotype mod mapping rule, and its ability to wrap its genome. GE is fully described in Chapter 2. While GE has been successfully applied to many application domains, see Section 2.5, to date, much of the work on GE has been performed using standard CFGs, or extensions thereof, with a few exceptions [102, 34, 182]. The goal of this thesis is to explore the utility of tree-adjoining grammars (TAG), a more powerful grammar formalism than CFGs, as part of the representation for GE.

Tree-Adjoining Grammars

TAGs were first introduced by Joshi et al. [97] in the field of Natural Language Processing and Linguistics. TAGs are a tree generating system, constructing derivation trees by composing smaller elementary trees together, and have been shown to be more powerful than CFGs [96]. TAGs can also generate some context-sensitive languages [96, 99].

TAGs capture the linguistic property that complex sentences can be viewed as being composed of more simple sentences which have been subjected to appropriate deformations [97], more directly representing the structure of natural language than is possible with CFGs. This has the effect of shortening the edit distances between the derivation trees of similar sentences. This property is inherent to TAGs and is a result of the non-fixed arity property of their derivation trees. TAGs have recently been successfully applied to the field of EC, and in particular GP [78, 85, 1].

This thesis proposes to explore the use of TAGs as part of the representation for GE. While the increased generative power of TAGs might be beneficial when used as a representation for search, TAGs can also help address a problem found in the current GE representation; the problem of genotypes mapping to unfeasible, or invalid, phenotypes.

1.2 Aim of Thesis

The aim of this thesis is to explore the use of TAGs for GE. TAGs are, generatively, more powerful than the CFGs which are traditionally used for GE. The intention of this thesis is to incorporate TAGs into the GE representation, and to investigate both the suitability of TAGs for use as part of a representation for GE, as well as the effect TAGs have on the ability of the algorithm to search effectively. In addition to being more powerful than CFGs, TAGs have a non-fixed arity property which will be explored for use with GE, specifically in the context of developmental evolution. The long term aim of this work is to add to the understanding of search using complex systems such as GE, and how the form of the grammar affects this.

1.2.1 Research Questions

The core aim of this thesis is to explore how using TAGs as part of this representation affects GE's ability to search, and to better understand search using the GE representation. To fulfil this aim a number of research questions are proposed:

- *Can TAGs be used, in conjunction with a linear genotype, as an effective representation for GE?*
- *How does a TAG-based GE representation behave differently than the canonical CFG representation used by GE?*
- *Do the biases introduced by converting CFGs to TAGs affect GE?*
- *Can any properties of TAGs be exploited in order to enhance GE?*

1.2.2 Objectives of Thesis

In order to address the aims and questions specified in this chapter, multiple objectives are presented:

1. Survey the state of the art for TAGs in EC.
2. Develop and implement a TAG-based representation for GE.
3. Perform analysis of the developed representation in terms of performance.
4. Analyse the differences in behaviour of the two representations.
5. Identify and examine biases present in the TAG-based representation.
6. Identify properties of the representation that might be used to explore extensions to GE.

1.3 Contributions

A number of works have been published in the completion of this thesis. These publications are listed on page xii. The main contributions of the exploration of TAGs for GE are outlined here in order of appearance:

Literature review

A survey of the previous literature on the use of TAGs in both GP and the extended EC field is presented in Chapter 3.

Novel TAG-based representation for GE

Section 5 presents a novel extension of the GE algorithm, tree-adjoining grammatical evolution (TAGE), which allows the use of TAGs with the GE linear chromosome.

The mapping process is described in full, along with the effects of the standard GE variation operations on this novel representation.

Analysis of TAGE performance

A comparison of performance between canonical GE and TAGE is presented in Chapter 6. It is shown that TAGE performs better than GE. However, differences between the two representations are identified, and when addressed, GE can perform as well, if not better than TAGE on some of the experiments. Separately, it is noted that while the best individuals observed in TAGE populations can be better than those in GE, the fitness of TAGE populations, on average, are worse than those of GE.

Novel Analysis of search landscapes

Chapter 7 examines the mutation search spaces for both TAGE and GE. Landscapes of regions of these spaces are visualised using heat maps to remove the visual clutter present when using a traditional graph-based representation. It is shown that the TAGE representation is more connected than GE.

Method of initialisation for identical initial GE and TAGE populations

When comparing different representations, it can be difficult to account for the initial position in the search space. Section 8.2 presents a method of generating well distributed initial populations which can be used for both GE and TAGE.

Identification and analysis of biases in TAGE representation

Chapter 8 explores language biases in both the TAGE representation and the transformed TAGs from commonly used CFGs. Methods for mitigating these biases are presented. It is shown that these biases modify the distribution of generated sentences. This can also affect the performance of the algorithm but is problem specific.

Literature Review

Section 4.1 provides a review of artificial gene-regulatory network models. In particular, a survey of previous work utilising these networks as representation for EC is presented.

Novel evolutionary developmental extension of TAGE

Section 9.1 presents a developmental extension of TAGE (DTAGE) where an artificial gene regulatory network (GRN) model is incorporated into the TAGE pipeline. This extension takes advantage of the feasibility property of TAGs, allowing valid phenotypes with very few codons. By augmenting the mapping process with a GRN, the complex dynamics that come about from this model can be taken advantage of by TAGE, and the GRN model can be utilised for the diverse problem classes that grammar-based GP can be applied to. Moreover, this extension of the TAGE mapping process enables it to be influenced by the problem state, or environment, providing individuals with the ability to express different phenotypic effects depending on the environment in which they find themselves.

Publications

Each chapter in the Exploration section of this thesis is based upon a published work. These works are listed here in order of publication:

1. Eoin Murphy, Michael O'Neill, Edgar Galvan-Lopez, and Anthony Brabazon. **Tree-adjunct grammatical evolution.** In *2010 IEEE World Congress on Computational Intelligence*, pages 4449–4456, Barcelona, Spain, 18-23 July 2010. IEEE Computational Intelligence Society, IEEE Press. doi: 10.1109/CEC.2010.5586497.
2. Eoin Murphy, Michael O'Neill, and Anthony Brabazon. **Examining mutation landscapes in grammar based genetic programming.** In Sara Silva,

James A. Foster, Miguel Nicolau, Mario Giacobini, and Penousal Machado, editors, *Proceedings of the 14th European Conference on Genetic Programming, EuroGP 2011*, volume 6621 of *LNCS*, pages 130–141, Turin, Italy, 27-29 April 2011. Springer Verlag. doi: 10.1007/978-3-642-20407-4_12. **Best paper**.

3. Eoin Murphy, Erik Hemberg, Miguel Nicolau, Michael O'Neill, and Anthony Brabazon. **Grammar bias and initialisation in grammar based genetic programming.** In Alberto Moraglio, Sara Silva, Krzysztof Krawiec, Penousal Machado, and Carlos Cotta, editors, *Proceedings of the 15th European Conference on Genetic Programming, EuroGP 2012*, volume 7244 of *LNCS*, pages 85–96, Malaga, Spain, 11-13 April 2012. Springer Verlag. doi: 10.1007/978-3-642-29139-5_8.

4. Eoin Murphy, Miguel Nicolau, Erik Hemberg, Michael O'Neill, and Anthony Brabazon. **Differential gene expression with tree-adjunct grammars.** In Carlos A. Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone, editors, *Parallel Problem Solving from Nature, PPSN XII (part 1)*, volume 7491 of *Lecture Notes in Computer Science*, pages 377–386, Taormina, Italy, September 1-5 2012. Springer. doi: 10.1007/978-3-642-32937-1_38.

1.4 Limitations

This work explores the use of TAGs as part of the representation for GE. As a novel representation is developed, some limitations must be considered to progress the study. Firstly, the investigations of this thesis concentrate on a single developed representation. While many other methods of incorporating TAGs into the GE representation may exist, this study exclusively considers the approach presented in Chapter 5.

Additionally, in running the GE algorithm, no exhaustive search of all possible operator combinations, or the possible parameter values is performed. Standard values are used throughout all experiments performed in Chapters 4-7. Furthermore, to aid in the analysis of the change in representation, only well known benchmark problems from classic GP literature are used.

Moreover, TAGs which are automatically generated from the CFGs used by canonical GE are exclusively utilised during the experiments in this thesis. By using these TAGs common languages can be guaranteed. By using user created TAGs, alternate biases could be introduced into the system, affecting the search spaces, and hence the comparison of representations.

1.5 Thesis Summary

Chapter 2 describes the GE algorithm. A definition of CFGs is provided and the genotype-phenotype mapping process is described in detail. The other operations used by the GE algorithm are also described.

Chapter 3 opens with a formal definition of TAGs, and goes on to provide a comprehensive survey of related work involving TAGs in EC. Particular attention is payed to the use of TAGs in the field of GP.

Chapter 4 is concerned with developmental evolution and how it has been used in the field of EC. A survey of research is provided, paying particular attention to gene regulatory networks.

Chapter 5 presents, TAGE, an extension of GE which incorporates TAGs into the representation. TAGE and its components are discussed in detail. The effect that this change in representation has on the behaviour of the variation operators is also examined, along with some limitations of the new representation.

Chapter 6 compares TAGE with standard GE in terms of their ability to solve a number of benchmark problems. It is initially shown that TAGE outperforms GE on all problems. A number of representational differences are identified, and when addressed, GE is capable of performing as well as, and indeed, better than TAGE on some of the experiments. While the best individuals in TAGE populations are often shown to be better than those of GE populations, the average fitness of TAGE populations are worse than those of GE.

Chapter 7 explores the difference in connectivity of the search spaces between TAGE and GE. A novel method of rendering search landscapes using heat maps is presented. It is shown that, for single mutation events, on a number of typical grammars, TAGE search spaces are much more densely connected than those of GE.

Chapter 8 examines changes in the biases inherent to the representation when transforming from CFG to TAG. While the original CFG and the produced TAG are equivalent, i.e., no exclusion biases are introduced during the transformation, the preferential language biases in the system are affected. The distribution of randomly generated derivation trees, and hence, phenotypes are examined and it is shown that initialisation can be affected. Methods for improving, and indeed, equalising the distributions of initial individuals are presented. Furthermore, additional biases are identified in the TAGE representation. These biases are also examined in how they affect the distribution of derivation trees, as well as how the can affect the algorithm's ability to perform.

Chapter 9 introduces DTAGE, a novel evolutionary developmental extension to TAGE using an artificial GRN model. Initially, related research is presented on using GRNs for EC, and an overview of the GRN model used by this thesis is provided. The developmental extension to TAGE is then described. DTAGE exploits the feasibility property of the TAGs, with the GRN embedded into the mapping process. This enables TAGE mapping to make use of the complex dynamics of the GRN model, while grammar-based approach enables the model to be used on a greater range of problem classes. This extension enables feedback from the problem environment which influences the mapping process, allowing the state of the environment to affect an individual's phenotypic effect. The method is tested on the benchmark inverted pendulum problem, and properties of the extension are discussed.

Chapter 10 concludes the thesis by giving an overview of the work presented throughout, and a summary of the main contributions. It also suggests and outlines future interesting work which may be pursued on the use of TAGs for GE.

Part I

Background

Chapter 2

Grammatical Evolution

It is important that the approach used throughout this thesis is described in full. As such, this chapter is dedicated to describing Grammatical Evolution [163, 172, 173, 42] (GE), a form of GBGP, in detail. GE makes use of aspects of Darwinian natural selection, classical genetics and molecular biology combined with the generative power of grammar formalisms, and is considered to be one of the most widely applied GP methods today [133].

This chapter provides an introduction to the GE algorithm in Section 2.1. The grammar type used by GE, Context-Free Grammars (CFG), is then defined in Section 2.2. Following this, the canonical GE algorithm itself is described in Section 2.3, along with the genotype-phenotype mapping process, which is central to GE, in Section 2.3.1. The individual components of the GE algorithm are then described separately. First, population initialisation is outlined in Section 2.4.1, with selection and replacement methods in Section 2.4.2 and 2.4.3 respectively. The variation operations used with GE are described in Section 2.4.4. Examples of different problem domains that GE has been applied to are given in Section 2.5, along with a list of implementations of the algorithm. Section 2.6 concludes with a summary of the chapter.

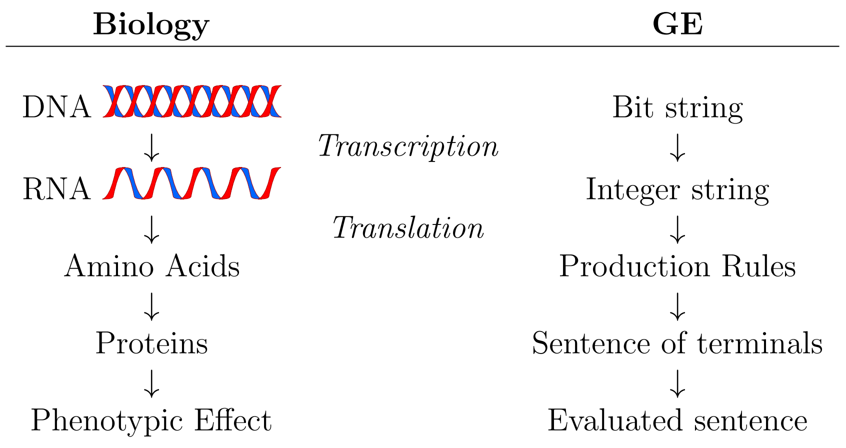


Figure 2.1: Analogy between molecular biology and the GE mapping process.

2.1 Overview

GE is a variable length, linear form of GP. Rather than evolving program trees directly as is done in GP, GE makes use of a variable length GA as the search engine. GE's representation consists of a bit or integer string (the genotype), upon which the GA searches, combined with a grammar. A mapping rule is used to map each evolved bit string to a sentence generated from the grammar (the phenotype). That sentence can then be evaluated. This mapping, the genotype to phenotype mapping, was inspired by gene expression in classical genetics (see Figure 2.1).

The grammar, which is usually a CFG written in Backus-Naur form, enables GE to define and modify the legal expressions of an arbitrary computer language. The grammar also enables GE to modify the structure of the expressions generated, biasing search towards particular structures, something that is not trivial for other forms of GP. Moreover, the grammar allows problem specific domain knowledge to be embedded within the generative process. An example of a GE grammar, genotype and resulting derivation tree are given in Figure 2.2.

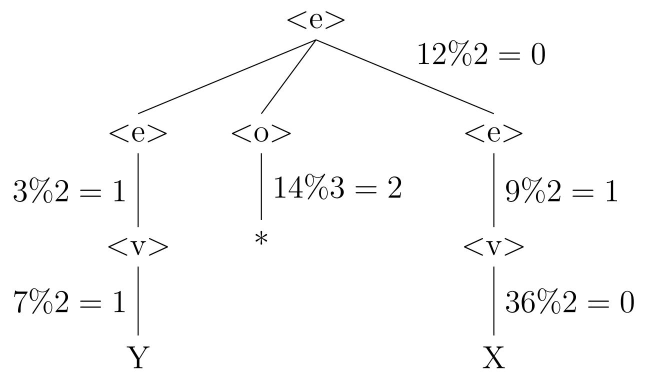
In addition, the separation of the genotype from the phenotype in GE allows GP style genetic operations to be applied to the interim tree-based derivation tree representation,

```
<e>:= <e><o><e>      (0)
    | <v>            (1)
<o>:= +              (0)
    | -              (1)
    | *              (2)
<v>:= X              (0)
    | Y              (1)
```

(a) A sample context-free grammar for use in symbolic regression. The grammar consists of three non-terminal symbols `<e>`, `<o>` and `<v>`. All other symbols are terminal symbols. There is a set of production rules for each non-terminal symbol. Each production rule, separated by |, is indexed, incrementing from zero. The indices are shown for clarity.

12	3	7	14	9	36	14

(b) A short sample GE chromosome consisting of a list of integers. Each integer is known as a codon.



(c) An example derivation tree created from the grammar and chromosome seen in Figure 2.2. The edges of the tree, from parent node to children, represent production rule choices. Each edge is labelled with the instance of the mapping function used to select that production rule.

Figure 2.2: An example GE grammar, chromosome and derivation tree.

as well as GA style operations to be applied to the linear genotype, extending GE's search capabilities beyond those of GP. The grammar type used by GE, CFGs, are formally described in the next section.

2.2 Context-Free Grammar

In formal language theory, a grammar is a generative system for the creation of the set of strings for a particular language. Grammars make use of a set of rules which define the structure of the words and sentences of a language. These rules are used to construct syntactically correct strings from that language. The symbols making up the alphabet of the language are known as *terminal symbols*. Another set of symbols, *non-terminal symbols*, are involved in the construction of strings but do not appear in the final strings themselves. The rules are used to transform strings containing both types of symbols into string composed entirely of terminal symbols in a process known as *derivation*.

The original and most commonly used grammar type with GE is the CFG. A CFG [69, p. 18] is a type-2 formal grammar in the Chomsky hierarchy, and is defined as follows.

Definition 1 (Context-Free Grammar) A CFG, G, can be defined as a quadtuple

$$G = (N, \Sigma, P, S)$$

where:

- N is a finite non-empty set of non-terminal symbols;
- Σ is a finite non-empty set of terminal symbols such that $N \cap \Sigma = \emptyset$. Σ^* denotes the set of all finite-length strings constructed from Σ, and V^* denotes the set all finite-length strings constructed from $N \cup \Sigma$ - both Σ^* and V^* contain the empty string;

- P is the set of production rules, where $P \subseteq N \times V^*$;
- S is the start symbol where $S \in N$. □

Production rules in CFGs are of the form

$$A \mapsto \alpha$$

where $A \in N$, a non-terminal symbol, and $\alpha \in V^*$, a non-empty string consisting of non-terminal and terminal symbols, known as an *expression*. This particular production rule denotes that the non-terminal symbol A *generates*, or may be *rewritten* as, α. This can occur regardless of the context in which it A appears, and it is for this property that these grammars are known as *context free*. The following definition of generation, or rewriting, is used [69, p. 14].

Definition 2 (Generation) Let $G = (N, \Sigma, P, S)$ be a CFG and let $\alpha', \beta' \in V^*$. α' directly generates β', written $\alpha' \Rightarrow \beta'$, if there exist $\alpha_1, \alpha_2, \alpha, \beta \in V^*$, such that $\alpha' = \alpha_1\alpha\alpha_2$, $\beta' = \alpha_1\beta\alpha_2$ and $\alpha \mapsto \beta \in P$. The reflexive-transitive closure of $\Rightarrow$ is written as $\overset{*}{\Rightarrow}$. □

The reflexive-transitive closure of $\Rightarrow$, $\overset{*}{\Rightarrow}$, is a multi-step generation, i.e., multiple applications of the $\Rightarrow$ operation. An operator is closed on a set when the application of that operator results in another member of that set. In the given definition, both $\alpha', \beta' \in V^*$, therefore V^* is closed under $\Rightarrow$.

Definition 3 (Sentential Form) Let $G = (N, \Sigma, P, S)$ be a CFG. The set of *sentential forms* of G, $S(G)$ is:

$$S(G) = \{\alpha \in V^* | \ S \overset{*}{\Rightarrow} \alpha\}$$

The set $S(G)$ is considered to be anything derivable from S. As $\alpha \in V^*$, α can contain non-terminal symbols also. However, if $\alpha \in \Sigma^*$, it is then known as a sentence. The set of

```
<A> := a<b>c
<b> := b | b<b>
```

Figure 2.3: An example grammar written in BNF which generates the set of strings $\{ab^nc \mid n > 0\}$.

sentences is the language generated by the grammar. All elements of $S(G)$ can generate sentences.

Definition 4 (Language) Let $G = (N, \Sigma, P, S)$ be a CFG. The language of G, $L(G)$ is the set:

$$L(G) = S(G) \cap \Sigma^* = \{\omega \in \Sigma^* |\ S \overset{*}{\Rightarrow} \omega\}$$

The preceding definitions are as given by Harrison [69].

The most common form in which CFGs are presented is the Backus-Naur form (BNF) [3]. An example of a CFG written in BNF is presented in Figure 2.3. Rules in BNF retain the form of a single non-terminal symbol on the left hand side (LHS), with an expression on the right (RHS). The LHS and RHS are separated by `::=`, which denotes that the LHS should generate the RHS. Non-terminal symbols in BNF are denoted using `<angle-brackets>`, and rules that share a common LHS are grouped using `|`. Figure 2.3 shows the previous example production rule rewritten in BNF, with α replaced with the string `a<b>c`, along with supplementary rules for the non-terminal symbol `<b>`.

2.3 Grammatical Evolution Algorithm

This section provides an overview of the GE algorithm, as portrayed in Figure 2.4, as well as presenting a detailed description the genotype-phenotype mapping process.

The GE algorithm, an extension of a standard GA, operates much like any other EA, with one main difference - the genotype to phenotype mapping. In order to describe GE a number of definitions are first required.

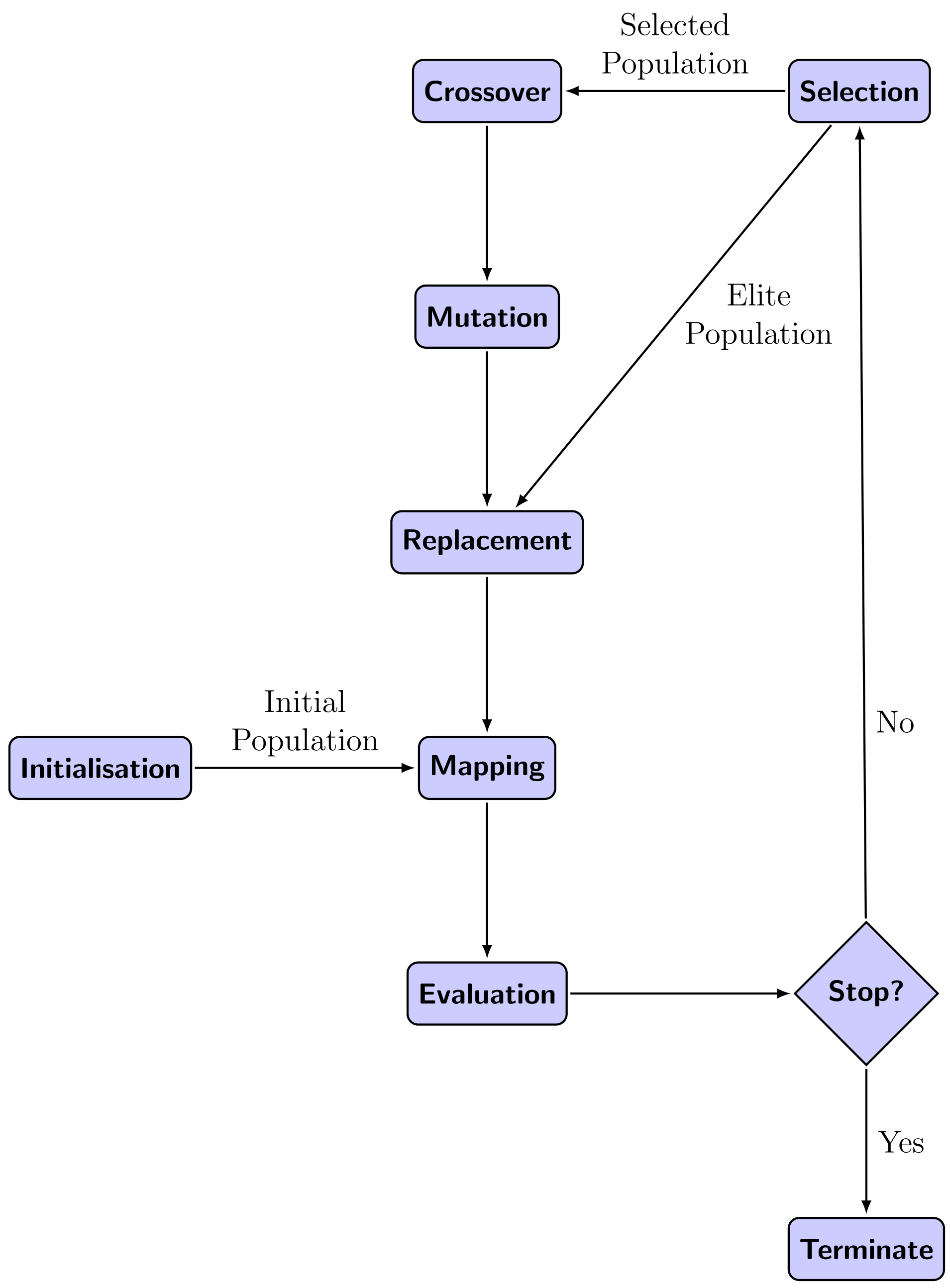


Figure 2.4: The component flow graph of the GE Algorithm.

Definition 5 (Codon) A codon is the smallest unit of representation. Codons in GE are comprised of a number of bits, usually 8 or 32 bits. A GE codon bit sequence can be expressed as an integer value to be used with the GE mapping function. □

Definition 6 (Chromosome) The GE chromosome used in this study is a string of integer values, or codons. □

Integer chromosomes are used in this thesis as Hugosson et al. [94] show that the integer genotype representation holds an advantage over other binary representations in terms of the performance.

Definition 7 (Genotype) A genotype is the base encoding for an individual in the population. Variation operations such as mutation and crossover are applied to the genotype. A chromosome is used as the genotype for individuals in GE. □

Definition 8 (Phenotype) The final representation of an individual, through which the individual's fitness may be measured. In GE, the phenotype is a sentence from the language defined by the grammar. The phenotype is mapped from the genotype using the grammar. □

Definition 9 (Derivation Tree) A derivation tree is a tree-based representation of the complete derivation of S into a sentence from the language. That is to say, it is a tree-based representation of $S \overset{*}{\Rightarrow}$ phenotype. Derivation trees are used as an intermediate state of representation for individuals when mapping from genotype to phenotype. Rooted with S, the children of each non-terminal node in the tree represent a rewriting of that node by means of some production rule. The codons from the genotype direct the selection of production rules. A complete derivation tree results in terminal nodes only being present along the frontier of that tree (leaf nodes). These terminal symbols, when read from left to right combine to form a valid sentence from the language, the phenotype. □

Execution of the GE algorithm, as portrayed in Figure 2.4, proceeds as follows. An initial population of individuals is generated. The chromosomes of each individual are mapped to derivation trees. From the leaf nodes of these trees, the phenotype of each individual is extracted. These sentences are evaluated by a fitness function in order to ascertain the fitness of each individual. If a specified target fitness has been achieved by an individual then the algorithm halts. Otherwise, a selection mechanism, based on fitness, is used to select individuals to be modified by variation operations, creating a new population. This population is then mapped and evaluated in the following generation. The process continues until one of two termination criteria has been achieved, either that the target fitness is reached or a specified number of generations, or iterations of the algorithm, have occurred.

The genotype-phenotype mapping process is described in detail in the following section. The remaining components of the algorithm are described separately in Section 2.4.

2.3.1 Genotype-Phenotype Mapping

The genotype-phenotype mapping process in GE maps a chromosome to a syntactically correct sentence. This is a deterministic process, given a specific genotype, non-terminal symbols are expanded using codons by selecting production rules from the grammar, constructing a specific phenotype.

An example genotype to phenotype mapping is given below in Ex. 1. The grammar used for the following example, along with the chromosome and resulting derivation tree can be seen in Figure 2.2. The entire derivation sequence is given in Figure 2.5. Pseudo-code for the mapping can be seen in Algorithm 2.1.

Example 1 (GE Mapping Example) Mapping begins with the start symbol S which is usually the first symbol declared in the grammar. In this case the start symbol is `<e>`.

Algorithm 2.1 GE Derivation - Mapping from genotype to phenotype using a grammar, G, and a chromosome, C. A derivation tree is constructed by expanding non-terminal leaf nodes in a depth first manner. A *maxWraps* parameter is used to restrict chromosomal wrapping. The phenotype is extracted from the leaf nodes of the completed tree.

```
Require: $G = \{N, \Sigma, P, S\}$ {A CFG}
Require: $C$ {A chromosome}
Require: $maxWraps$ {The maximum number of wrapping operations}
  $wraps \leftarrow 0$
  $tree.root \leftarrow S$
  while $tree$.hasNTLeafNode and ($C$.size$> 0$ or $wraps \leq maxWraps$) do
    if $C$.size$= 0$ then
      $C$.reset
      $wraps \leftarrow wraps + 1$
    $node \leftarrow tree$.getDepthFirstNTLeaf
    $lhs \leftarrow node$.getSymbol
    $rules \leftarrow P$.getRules($lhs$)
    $i \leftarrow 0$
    if $rules$.size$> 1$ then
      $i \leftarrow C$.removeFirst mod $rules$.size
    for all $S_j$ in $rules$.get($i$).reverse do
      $node$.addChildNode($S_j$)
  $phenotype \leftarrow \epsilon$
  for all $node_i$ in $tree$.getLeafNodes do
    $phenotype \leftarrow phenotype + node_i$.getSymbol
  return $phenotype$
```

The GE mapping procedure chooses a non-terminal symbol to expand, counts the number of production rules for that symbol, and if needed, reads a codon to select a production rule. If there is only one production rule, i.e., there is no choice, then a codon is not read, otherwise the following mapping function is used

$$\mathit{codonvalue} \bmod ||\mathit{productions}|| = \mathit{productionIndex}.$$

In this case the symbol `<e>` has two production rules

```
                0          1
<e>:= <e><o><e> | <v>
```

The first codon, or integer value, is read from the chromosome, `12`. The mapping function is used to choose a production rule, `12 mod 2 = 0`, therefore we chose the zero-th rule and `<e>` is expanded to `<e><o><e>`.

This expansion forms a partial derivation tree with the start symbol as the root, attaching each of the symbols from the chosen rule as child nodes to this root (Figure 2.5b). The next symbol to expand is the first non-terminal leaf node encountered while traversing the derivation tree in a depth first manner. In this case, this is the left-most `<e>` in the tree. Again, `<e>` has two production rules, so a codon is read. The next codon has a value of `3`, `3 mod 2 = 1`, expanding this `<e>` to `<v>` and growing the tree further (Figure 2.5c).

The next symbol to expand is the newly added `<v>` which has two possible production rules

```
      0   1
<v>:= X | Y.
```

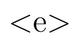
The next codon is read and has a value of `7`, `7 mod 2 = 1`, so the rule at index 1 is chosen: `Y` (Figure 2.5d). The next non-terminal node is chosen in order, `<o>`, and the another codon is read to select a production rule from the grammar. The process continues until some termination criterion has been reached. The complete derivation sequence for this example is shown in Figure 2.5. □

Mapping Termination

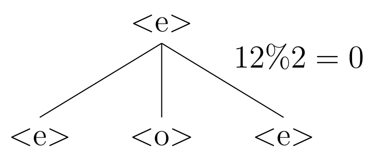
The mapping process continues until either there are no non-terminal leaf nodes remaining, or the end of the chromosome has been reached. If the former occurs then mapping has terminated successfully and the produced phenotype can be evaluated. Otherwise, if there are no codons remaining and the derivation tree has leaf nodes labelled with non-terminal symbols, then mapping has not terminated successfully and there are a number of ways to proceed.

The first approach is that codons may be reused in an attempt to complete mapping. Inspired by the gene-overlapping phenomenon that is observed in many organisms [10], *wrapping* the chromosome results in returning to the beginning of the genotype to begin reading the codons from left to right again. Wrapping has an interesting consequence in that the same codon can now influence the resulting phenotype in multiple ways, introducing additional functional dependencies within the chromosome. Wrapping has been shown to work on a number of problems [173].

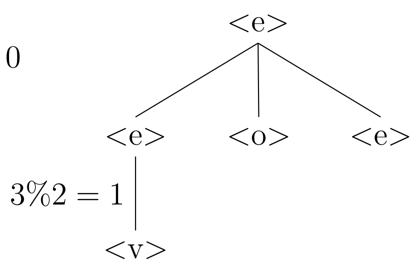
The second approach, and perhaps the most trivial, is to mark the individual as *invalid*. If after a specified number of wrappings, or if wrapping is not in use, an individual has not completely mapped, it is then marked as an invalid individual and assigned a poor fitness. This decreases the probability that the individual will pass any information on to the next generation due to selection and replacement pressure. An important point about wrapping, recently presented in work by Nicolau et al. [157], is that wrapping alone does not guarantee that mapping will terminate. If, directly upon wrapping the chromosome, the non-terminal symbol being expanded has been expanded by this codon before, then the mapping process will cycle indefinitely.



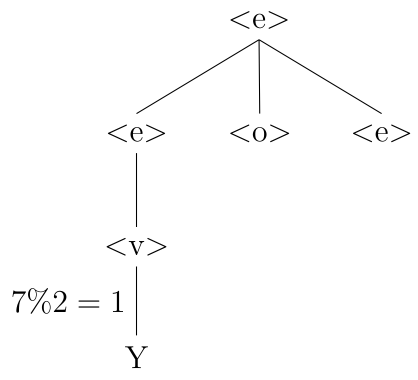
(a) Start symbol



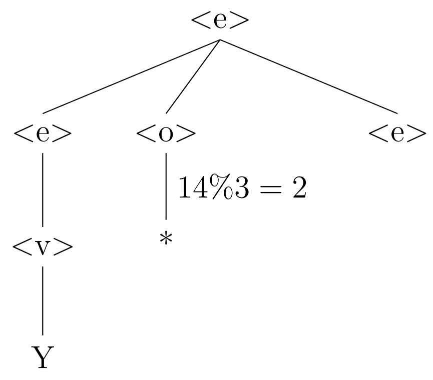
(b) Expansion to <e><o><e>



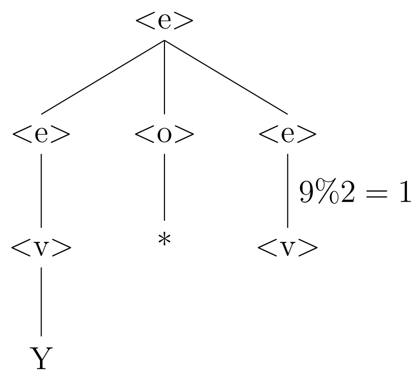
(c) Expansion of <e> to <v>



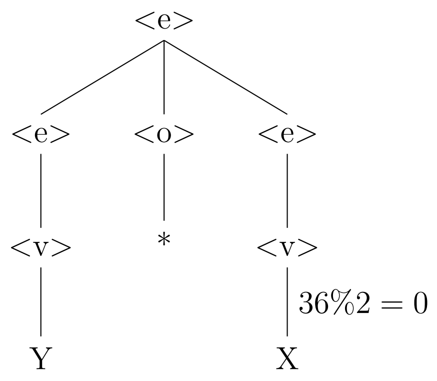
(d) Expansion of <v> to Y



(e) Expansion of <o> to *



(f) Expansion of <e> to <v>



(g) Expansion of <v> to X

Figure 2.5: An example derivation sequence created from the grammar and chromosome in Figure 2.2. The sub-figures show the derivation tree at each stage of derivation. This results in the phenotype, Y * X, which is extracted from the leaf nodes of the complete derivation tree.

Other approaches for dealing with non-mapping individuals include attempting to prevent invalids from being generated. Hemberg and O'Reilly [74], when generating HEML grammars for their Genr8 application, impose a depth limit, `max_depth`, on the derivation trees, such that when `max_depth - 1` has been reached, production rules containing only terminal symbols may be used to complete the tree. This approach was originally used to prevent very large trees from being generated, however, it also prevents the trees from expanding indefinitely and generating invalids. Additionally, it is possible to repair invalid individuals. Paterson [184] achieves this by making use of default expansion rules. If mapping terminates due to there being an insufficient number of codons, then the application of the default expansion rules to the remaining non-terminal leaf nodes results in a complete derivation tree, and a feasible phenotype.

Mapping Variations

While standard GE genotype to phenotype mapping chooses which node to expand in a depth first manner, there is on-going study into variations of this method.

πGE O'Neill et al. [162] proposed πGE [18, 54, 47, 52] which enables the expansion order to be evolved. A list of non-terminal leaf nodes is kept, and if there is a choice of non-terminal nodes to expand, then a codon from the chromosome is used to decide using the mapping function. πGE has been successfully applied to some problems [18, 54].

More recently Fagan et al. [48, 49] have expanded upon πGE, looking at many different expansion orders and how the order of expansion can affect the performance of GE. In addition, Nuez and Watt [161] have looked at embedding domain knowledge into the mapping algorithm itself. Working on a depth limited problem, the MAX problem, this depth limit is incorporated into the mapping preventing the limit being broken at the mapping level, generating only valid trees.

The Bucket Rule Keijzer et al. [104] present an alternative to the standard GE mod-based mapping rule, the Bucket Rule. This rule addresses the linkage found between production rules of different non-terminal symbols when mapping using the same codon with the mod rule. The adoption of this rule was shown to be beneficial.

Chorus [198, 2] Chorus is a position independent encoding system for grammar-based EA. The system is based on the manner in which genes produce proteins that regulate the metabolic pathways of the cell. The phenotype is the behaviour of the cell's metabolism, in this case, corresponding to the development of a computer program. In this procedure, the actual protein, or grammar rule, encoded by a gene is the same, regardless of the position of the gene within the genome. When mapping non-terminal symbols in a depth-first manner, a concentration, or count table is consulted for the relevant production rule with the highest concentration. That production rule is then used for the expansion in question, and the count at its index in the table is decremented. If a tie occurs, codons from the chromosome are read. Each codon, when read, is modded by the *total* number of production rules in the grammar and that index in the table is incremented, continuing until the tie is resolved.

GAuGE [153, 43] Genetic Algorithms using Grammatical Evolution is a position independent approach to evolving GA-type genotypes using a mapping rule similar to GE. Bit string genotypes, whose lengths are problem dependent, are converted into pairs of integers. The first integer value is used to chose a position in the output string by using the mod rule in conjunction with the number of positions in the output. The second codon is then used to assign a value at that position, also using the mod rule, this time with the maximum value allowed in the output string, i.e., `c mod 2` if the problem is binary in form.

GE^2 [174] Grammatical Evolution by Grammatical Evolution is a diploid, meta-grammar approach. A meta-grammar is a grammar which generates grammars. The first chromosome is mapped by a meta-grammar producing a new grammar. This produced grammar is then used to map the second chromosome into the phenotype string. Meta-grammars are used in an effort to generate grammars which successfully restrict the phenotypic search space for each individual. Crossover operates on homologous chromosomes.

Grammar Variations and Examinations

In addition to the standard CFG, a number of grammar extensions, as well as different grammar types, have been used in conjunction with GE.

Adaptive logic programming grammars Keijzer et al. [102] apply a GE approach to the derivation of non-deterministic logic programs for the Prolog programming language.

Attribute grammars Cleary and O'Neill [34] enhance GE by means of attribute grammars. Attribute grammars allow GE to maintain some context-sensitive and semantic information. When applied to the Knapsack problem it is shown that attribute grammars provide an improvement over standard GE.

Christiansen grammars Ortega et al. [182] make use of Christiansen grammars, an extension of attribute grammars where the first attribute of each symbol in the grammar contains a set of production rules applicable to that symbol. This set can be recomputed during derivation.

L-system grammars Hemberg and O'Reilly [74] use CFGs to evolve L-System grammars as part of a architectural surface design tool, while Harper [65] implements a dynamic parameterised L-System using a novel grammar extension with GE.

Automatically defined functions (ADFs) Harper and Blair [67, 68] introduce the ability to call ADFs in the phenotype by inserting new production rules in the grammar each time a function is defined during mapping. Separately, Hemberg et al. [72, 70] examine the use of the meta-grammar approach, GE^2. The meta-grammar is used to define a grammar of ADFs. A second chromosome is then mapped to a phenotype using this ADF grammar.

Module detection and grammar rewriting Swafford et al. [209] investigate different methods for detecting useful modules produced in GE derivation trees, and explores the potential of modifying the grammar to affect the bias of useful modules being utilised by the population [211, 210].

meta-Grammar GA [165] mGGA applies the GE^2 approach to GAs. mGGA utilises the meta-grammar approach to evolve modular grammars for the generation of binary string GA genotypes.

2.4 Search in GE

This section outlines the operations used by GE to perform search. While the search engine used by canonical GE can be described as a GA, the modular nature of GE, see Figure 2.6, allows this to be replaced by any other method of search. O'Neill and Brabazon [164, 178, 167], in their system Grammatical Swarm, use particle swarm optimisation in place of the standard GA for search in GE. In addition to this, O'Neill and Brabazon [166] apply differential evolutionary search (DE) to GE, in the form of Grammatical Differential Evolution (GDE). GDE makes use of DE's real valued vector representation, and associated variation, or perturbation, operators. In order to be used with the GE mapping function, the values in each vector are rounded to the closest integer value and used as codons for

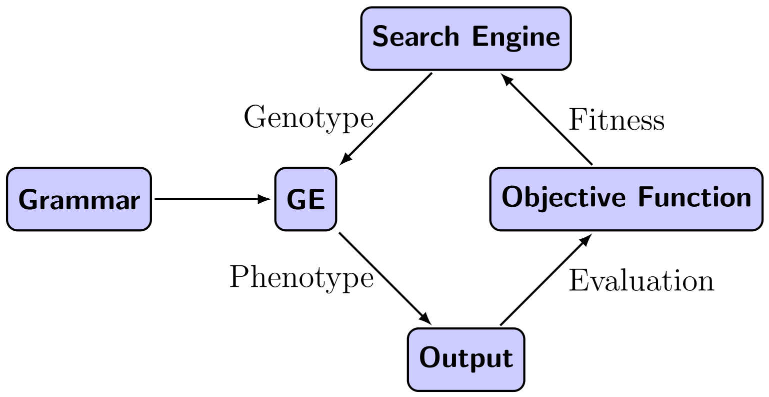


Figure 2.6: GE's modular design and flow

mapping.

This section goes on to describe the various components of the GE algorithm: population initialisation, selection, replacement, and variation operations.

2.4.1 Initialisation

The initial population can be generated using a number of different methods, the most straight forward of which is random initialisation. A chromosome length, l, is specified and l random integers are drawn from a uniform distribution for each chromosome. While this ensures an ideal distribution of initial codon values, depending on whether the grammar is explosive[1] and balanced[2], the distribution of derivation tree sizes, and hence the distribution of phenotypes in the initial population can be skewed [63].

The most common method of initialisation in GE is a form of Ramped Half and Half used in GP [113], otherwise known as Sensible Initialisation [196]. Sensible initialisation provides a greater spread of derivation tree shapes and sizes. Minimum and maximum

[1]An explosive grammar is a grammar where, if there is a choice between non-terminal and terminal nodes to expand, a non-terminal node is more likely to be selected [63].

[2]Harper [63] defines a GE grammar as being balanced if there is no non-terminal which is more likely than not to expand into multiples of the same non-terminal, and there is at least one non-terminal which will be reached in every parse of the grammar where the sum of the product of each of its production's recursive expansion factor and probability of being selected sums to 100%.

depths for initial derivation trees are specified, with the generated trees being evenly distributed within that range. A probability indicating the likelihood of using one method of tree generation over another, *full* or *grow*, is also provided. Beginning with the start symbol defined in the grammar, the full method initially chooses non-terminating production rules. These rules expand each branch of the derivation tree choosing expansions such that the maximum depth will not be exceeded. Similarly to full, the grow method chooses non-terminating production rules until an adequate depth has been reached. However, only a single branch must reach this depth. Once the depth has been reached, terminating production rules are chosen in a stochastic manner until the tree is complete. Chromosomes are generated retrospectively from the complete trees produced by these methods.

Other methods of initialisation include a variation of PTC2 [126] by Harper [63]. Harper highlights the importance of good initialisation and how bad initialisation can have a lasting effect. The modified PTC2 approach is a ramped method whereby non-terminal leaf nodes are chosen at random to be expanded. However, rather than the depth of the tree being limited, the number of non-terminal expansions is restricted. This gives a more even distribution of tree sizes and tree shapes than sensible initialisation.

Nicolau et al. [157] have also looked into including extra non-coding codons, otherwise known as artificial tails, at the end of sensibly initialised chromosomes (or indeed any method of initialisation where the chromosomes are generated retrospectively). Nicolau et al. show that these tails include terminating sequences, and by including the tails in the chromosomes, the probability that invalid individuals might be produced via variation operations is greatly reduced.

2.4.2 Selection

Selection is used to choose individuals from the current population. Variation operations are applied to these individuals in order to create new individuals. Two main selection mechanisms are used by GE, *fitness proportionate selection* and *tournament selection*.

In fitness proportionate selection, the probability of an individual being selected corresponds to the fitness of that individual in relation to the total fitness of other members of the population. The probability of an individual being selected, p_i is calculated as

$$p_i = \frac{f_i}{\sum_{j=0}^{N} f_j}$$

where f_j is the fitness of the j-th individual in the population and N is the size of the population. This selection mechanism is also known as *roulette selection*, as the probability of selection for each individual can be considered as a proportionately sized slice of the roulette wheel. As the ball spins around the wheel, there is a greater chance of the ball coming to a rest on a larger slice than a smaller one, and hence, the individuals with larger slices have a greater chance of being selected.

Tournament selection [137] compares the fitness of k individuals chosen in a stochastic manner, selecting the individual with the best fitness. This individual will be applied to variation operations and will pass along some of its information to the next generation. By modifying the value of k in relation to the population size, the level of selection pressure can be varied. Low values of k result in a low probability that any specific individual, e.g., very fit individuals, will end up in a tournament, lowering the selection pressure. As $\lim_{k \to N}$, where N is the population size, the probability increases that individuals with better fitness values will contend in each tournament, and as such, the selection pressure increases.

2.4.3 Replacement

Replacement is the mechanism which dictates the number of new individuals generated by means of variation operations, as well as which individuals, those in the current population as well as newly generated individuals, will be passed on to the next generation, or iteration of the algorithm. Two different approaches to replacement are used in GE, *generational* and *steady-state* [197].

The generational replacement mechanism replaces one population with another. N new individuals are generated from the current population using variation operations and are then used to replace that population. If *elitism* is being used, preserving the most fit individuals from one generation to the next, then n elites are copied from the current population before being replaced. The worst n individuals from the newly generated population are then removed, merging the preserved elites into the new population in their place.

Steady-state on the other hand, does not generate an entirely new population, but rather generates only n new individuals. Traditionally, n is quite low, with one or two new individuals being introduced into the population each iteration. While steady-state uses much less memory than generational replacement, it is more susceptible to premature convergence when using low values for n. Individuals have the opportunity to survive for many iterations as very few replacements occur, and as such, this can cause the population to converge towards its most fit members. Luke [127] discusses different approaches to address this problem such as randomly selecting individuals to be replaced rather than selecting the least fit individuals. This method is also known as *generation gap* replacement when large values of n are used. When $n = N$ then steady-state behaves the same as generational.

Another function performed by replacement in GE is to deal with the issue of *invalid* or non-mapping individuals in the population. Invalid individuals cannot be evaluated, and as such are given a default bad fitness value. Depending on the selection mechanism in use, if there is insufficient selection pressure, then invalid individuals can survive and be passed on to the next generation. Having invalid individuals in the population hinders the ability of the algorithm to search. To this end, replacement strategies are used to reclaim the invalid parts of the population. Invalid individuals may either be *a*) repaired by adding genetic material to the end of the chromosome in a effort to allow mapping to complete; *b*) repaired by using default expansion rules to complete the derivation tree, as was discussed in Section 2.3.1; *c*) or replaced by removing the invalid individuals from the population and replacing them either with one of their parent individuals or a new randomly initialised individual generated using the initialisation method.

2.4.4 Variation Operations

Variation operations are applied to existing genotypes in order to create new individuals, further exploring the search space. The variation operations commonly used with GE are similar to those used with GAs. However, there is a directional dependence inherent to GE's genotype-phenotype mapping, i.e., the function of a codon during mapping is dependent upon all of the codons previously read in that chromosome. As a result of this, modifications to a genotype can have a *ripple effect* on its corresponding phenotype [173, 103], i.e., while the mapping process from the beginning of the chromosome up to the mutated codon remains unchanged, mapping from that codon forward is affected. As a result of this dependence, ripple effects which occur due to modifications closer to the beginning of the chromosome are more destructive, affecting a greater proportion of the derivation tree, and hence, the phenotype. The two variation operations commonly used in GE are described below.

Crossover

Single point crossover [177], similar to GAs, exchanges sections of chromosome between two parents. A point along each parent chromosome is chosen and all codons from those points to the ends of their respective chromosomes are exchanged with the other parent.

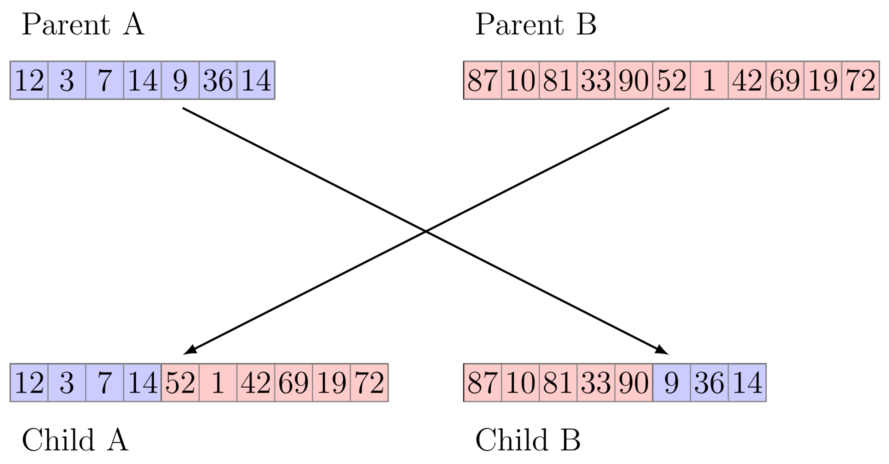


Figure 2.7: Single point crossover operation recombining two parent chromosomes into two child chromosomes.

This produces two new individuals each composed of one part from each parent. This can be seen in Figure 2.7.

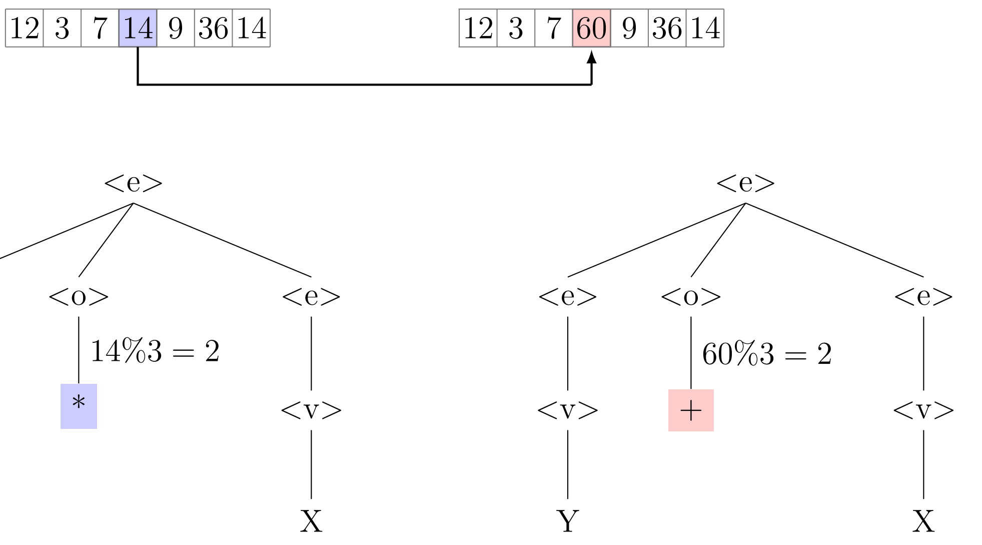
A useful extension to crossover operators is to limit the selection of crossover points to the region of the chromosome which encodes the phenotype. This limits *neutral crossover*, whereby crossover occurs in the non-phenotype encoding region of the chromosome.

A number of different crossover operations have also been used with GE. One extension of standard single point crossover uses a fixed point along both chromosomes. This can be useful if a fixed length representation is required. Harper and Blair [66] provide an overview of a number of existing crossover operations used by GE as well as introducing some new methods, including a structural preserving crossover operator. A two point crossover is proposed, which chooses initial points on each parent such that the selected codons expand the same non-terminal symbol. Another point is selected along each chromosome in the region following the initial points ensuring that only the codons required to fully expand those non-terminals are exchanged. This prevents any ripple effect from occurring. This method can be thought of as form sub-derivation tree crossover and was shown to provide substantial benefits over all other operators included in the study [66].

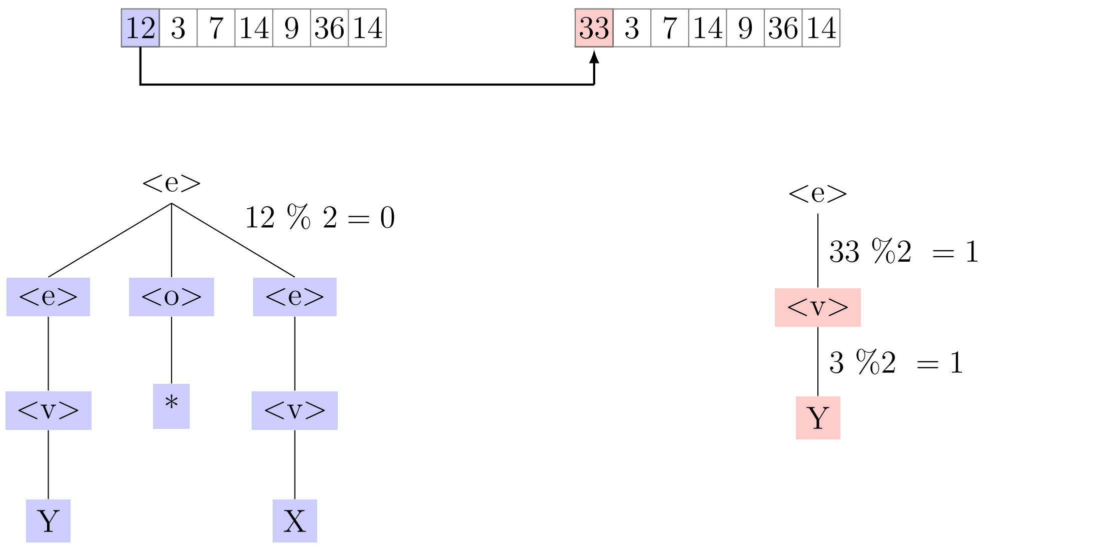
Nicolau and Dempsey [152] take a different approach, by extending the CFGs used, they allow the specification of crossover sites as non-terminal symbols within the grammar. When mapping, the index along the chromosome of the codons which are used to “expand” these crossover site symbols are saved within the genotype. It is at these codons only that crossover may occur. If an individual has no crossover sites saved in the genotype then it can not be used for crossover. This method is useful when crossover is only desired at specific locations within the phenotype. This approach has since been extended by [151] to allow two point crossover to occur between two crossover sites. Similar to the sub-derivation tree crossover by Harper and Blair [66], eliminating any ripple effect, but only allowing crossover to occur between user specified sub-trees.

Mutation

Mutation operators in GE are similar to those of GAs. The most commonly used operators are known as bit-flip, or integer-flip mutation, depending on the genotype representation being used. When using these operators, the value of each codon in a chromosome can be modified based upon a uniform probability distribution. The probability of a mutation event occurring to a single codon is a user-defined parameter. If a mutation occurs, a replacement integer value is drawn from a uniform distribution. An example of a mutation event affecting the generated phenotype is given in Figure 2.8.



(a) Mutation event where the mutated codon encodes a single terminal node. As the non-terminal nodes in the tree are uneffected no ripple effect is observed. The new phenotype is: Y + X.



(b) Mutation event where the entire tree is dependent on the selected codon. This causes a ripple effect in the tree, as all codons following the mutated codon are effected and a different tree is be derived. This highlights the high linear dependence can be inherent to GE chromosomes. The new phenotype is, Y.

Figure 2.8: Examples of the effect of single integer-flip mutation events on the GE mapping process. Multiple integer-flip events can occur each time that the operation is applied to a chromosome. The unmodified chromosomes and derivation trees on the left are mutated, resulting in the chromosomes and derivation trees on the right. The derivation nodes dependent upon the selected codons are highlighted pre and post mutation.

While the operations themselves are similar to those used with GAs, as the genotype is only one part of the representation, the effect of these operations can be different. Rothlauf and Oetzel [195] investigate the property of *locality* in the context of bit-flip mutation in GE. Locality is defined as the relationship between the magnitude of change to a genotype and the magnitude of the resulting change to the phenotype. High locality indicates that a small change in one, reflects a small change in the other, and is believed to be a useful property for local search around individuals of high fitness. Whereas low locality indicates that a small change in the genotype can result in a large change in the phenotype. While low locality is not useful for performing guided local search, it is useful for escaping from local optima, a property which Rothlauf and Oetzel show exists with mutation on the GE representation. It is shown that, often, neighbouring genotypes do not map to neighbouring phenotypes, making local search about individuals of high fitness difficult. It is revealed that this lower locality retards performance and that GE could benefit from mutation operations, or a modified representation, with higher locality.

Byrne et al. [30, 28] discover that the behaviour of integer-flip mutation in GE could be classified into two major behaviours. In order to test the contribution of each sub-behaviour, integer-flip mutation is decomposed into two independent mutation operations, *nodal mutation* and *structural mutation*. Nodal mutations affect codons which map to terminal symbols only, i.e., the leaf nodes of the derivation tree. Whereas structural mutation applies to those codons which map to non-terminal symbols, or internal nodes. These operations, when tested against standard integer mutation and sub-derivation tree mutation, display contradictory behaviours. Nodal is exploitative, refining existing derivation tree structures, with structural being more exploratory, creating variations of existing derivation trees.

Byrne et al. [32, 29] go on to examine the applicability of these mutation operations for use in user directed search in evolutionary design. In particular in the context of locality. Mutations with low locality, such as structural mutation, allow users to jump between designs of varying aesthetics, whereas high locality mutations enable users to refine designs which interest them.

Additionally, *plus minus mutation* is introduced by Hugosson et al. [94] in their examination of mutation and representations in GE. Plus minus mutation increments or decrements the current integer value of a codon by one, rather than replacing the codon with a randomly drawn value. This operator eliminates most forms of neutral mutation. Mutations which modify the phenotype while not affecting fitness are still possible. Hugosson et al. compare plus minus mutation with bit-flip and integer-flip mutation.

More recently, Fagan et al. [50, 51], inspired by how E.Coli bacteria behave when entering a new environment, explore the use of adaptive mutation rates with GE. A fitness reactive mutation operator is introduced, varying the mutation rate across generations. This operator uses a moving window to observe the best fitness in the population across generations. If a plateau in best fitness is detected, i.e., no change in fitness across the window, then it is suggested that the population has happened upon a local optima, and as such, the operator increases the mutation rate in an attempt to counteract any possible premature convergence, encouraging the population to explore further afield from this optima. It is shown that this fitness reactive mutation operator is a good replacement for standard mutation, as it performs as well as standard mutation with a parameter sweep to detect the optimal rate mutation.

2.5 Applications and Implementations

The use of a user defined grammar has allowed GE to be easily applied to many different problem domains. This section outlines a number of these domains. In addition, a list of GE implementations in various programming languages is also provided in Table 2.1.

Table 2.1: Implementations of GE in various programming languages. A more complete list of implementations is available at http://www.grammatical-evolution.org.

Name	**Language**	**URL**
CGE - GE in C	C	https://github.com/eoinomurchu/CGE
ECJ [46] - EC framework	Java	http://cs.gmu.edu/∼eclab/projects/ecj/
DRP - GE/GP hybrid framework	Ruby	http://drp.rubyforge.org
GERET - GE exploratory tool	Ruby	http://geret.org
GEVA [180] - GE framework	Java	http://ncra.ucd.ie/geva
libGE [155]	C++	http://bds.ul.ie/libGE/libGE/
ponyge [71] - Pony-sized GE	Python	https://code.google.com/p/ponyge
PyNeurGen - GE/Neural Network hybrid	Python	http://pyneurgen.sourceforge.net

In general, while studying some aspect of the algorithm, GE is applied to various classes of benchmark problems, e.g., symbolic and Boolean regression, classification and path finding problems [173]. GE has, however, also been applied to solving many other more practical problems, such as the discovery of caching algorithms [170, 171], generating test cases to aid in the validation of plagiarism detection systems [33], evolving digital circuits [37], mobile and ad hoc network intrusion detection [201, 218], and evolving weights and topologies of artificial neural networks [44] to name a few. GE has also been used to both improve the performance and optimise the energy usage of femtocell transmitters in corporate office environments [73].

A major field of problems to which GE has been applied is that of computational finance, including financial analysis and trading, with a number of books having been written on the topic [42, 19, 14]. From early on GE has been applied to problems such as the evolution of trading rules for different kinds of markets, including index [40] and foreign-exchange markets [23], amongst others [176, 24, 25]. Other problems include stock selection [131], efficient trading strategies [35, 36, 41], time-series prediction [22] and bond credit rating classification [17, 18, 20]. In addition, Brabazon and O'Neill have also applied GE to the analysis and detection of bankruptcy and corporate failure [21, 15, 16], while McGarraghy and Phelan [130, 188] address the problem of developing ordering policies which minimise overall supply chain cost.

Another field to which GE has been applied is bioinformatics. O'Neill et al. [179] applied GE to the task of recognising eukaryotic promoters, helping in the identification of genes. While Georgoulas et al. [56] made use of GE to classify foetal heart rates in an effort to improve the performance of electronic foetal monitoring, a method of detecting abnormal fetuses. GE has also been used by Motsinger et al. [138] to evolve neural networks for feature selection in genetic epidemiology to help in the prediction of complex diseases. More recently, Nicolau et al. [159] used GE to generate interpolating models for Net Ecosystem Exchange of carbon dioxide between the atmosphere and biosphere. Such data is often incomplete but is important in order to estimate annual carbon budgets.

Additional work has been done in the field of control algorithms, or agent control. An early application of GE was to the automatic programming of robots [183, 175], work that has continued more recently by Burbidge et al. [26]. Further to robots, work has been done on the generation of behaviours for artificial agents, e.g., Cui et al. [35, 36] use GE to evolve behaviours for agents in a simulated stock market. A broad area of work on artificial agents is that of the generation of behaviours for non-player characters in computer games. Galvan-Lopez et al. [55] evolve controllers for the Mrs. Pacman game and Harper [64] pitches evolved Robocode tank controllers against each other using a novel spatial co-evolution layered approach, obtaining comparable performance to human-coded controllers. Perez et al. [187, 186] take a different approach and evolve behaviour trees, in conjunction with the A* path-finding algorithm, to guide Mario through a variety of levels. This approach was found to be competitive with human-coded controllers using hard AI methods.

Moreover, the application of GE to design has been popular. O'Neill et al. [181] explore the use of shape grammars with GE for the generation of shelter designs. McDermott et al. [129] introduce a new kind of grammar specifically suited for the automatic generation of designs and compare this new grammar with shape grammars for a number of design tasks. Byrne [27] makes use of this grammar as part of an interactive evolution design tool for generating different structures, including the generation of structurally analysed, industry compliant, electricity pylon designs, as well as, bridge designs [32, 31]. GE has also been use as part of a surface design tool for architects, producing surfaces with an organic quality [74].

In terms of art, GE has been successfully used to evolve logos [168, 151], as well as the generation of 3D gait optimisation for horse model animation [145].

GE has also been applied to both the automatic and interactive generation of music. Reddin et al. [191] use GE for the automatic generation of short musical compositions, while Shao et al. [203] present a novel system for interactive generative music. Additionally, McDermott et al. [128] propose new user interface techniques to address limitations of interactive EC in the domain of sound synthesis, including slow evaluation, user fatigue, and the requirement for small populations.

2.6 Summary

This chapter describes the standard form of the GE algorithm. CFGs are formally defined and the mapping process from genotype to phenotype is outlined, with an example mapping provided. Population initialisation is described next, followed by selection and replacement mechanisms, and variation operations. A table of available implementations of the algorithm is provided and the chapter concludes by outlining some of the diverse problem domains that GE has been successfully applied to. This thesis extends the GE algorithm to make use of TAGs. The following chapter proceeds with a description of TAGs and a comprehensive survey of their use in EC. Subsequent to this, a discussion on developmental EC is provided.

Chapter 3

Tree-Adjoining Grammars

The purpose of this chapter is to provide an outline of TAGs and their use in the field of EAs. TAGs, which are described in detail in Section 3.1, have been used previously in GP in the form of TAG3P by Hoai and McKay [86] and later TAG3P+ [82, 78]. They have also been used for incremental and developmental evaluation [91, 87] and ant colony optimisation [1]. This thesis, however, presents a novel approach of combining TAGs with the GE algorithm, including its linear chromosome and genotype-phenotype mapping rule.

In preparation, this chapter proceeds with a definition of TAGs and discussion of some of the properties of TAGs in Section 3.1. This is followed by a review of the use of TAGs in EC in Section 3.2.

3.1 Tree-Adjoining Grammars

TAGs, a tree generating system previously known as tree-adjunct grammars, were first introduced by Joshi et al. [97]. TAGs were originally used in the field of Natural Language Processing and Linguistics with much success [114, 96]. More recently, however, TAGs have been applied to the field of EC [78, 85, 1]. A recent review of TAGs is provided by Joshi and Schabes [99], while a comprehensive review of TAG literature, in particular in the context of TAGs for GP, is given by Hoai [78].

A TAG is a grammar of trees. A tree, in this context, is a structural description of a sentence. A TAG is, therefore, a grammar of structural descriptions. There are two types of trees in a TAG: initial trees, also known as α or center trees, which are complete elementary structured sentences; and auxiliary trees, otherwise known as β or adjunct trees, which are used to modify structured sentences.

It is this property in part, of using structured objects as the elementary components of the grammar, that makes TAGs both interesting and powerful. By relating with the structural description of sentences, TAGs can manipulate sentences from one form to another more easily than other formalisms which relate directly with the set of strings. This enables TAGs to capture the linguistic property that complex sentences can be viewed as being composed of more simple sentences, which have been subjected to appropriate deformations [97].

This linguistic property is evident in TAGs as each intermediate step of derivation proceeds from one complete structured sentence to another complete structured sentence by means of a composition operation. This property has been coined *"feasibility"* by Hoai et al. [82]. For example, the sentence, "The fox jumps over the dog", becomes "The quick brown fox jumps over the lazy dog", simply by adding three nodes to the frontier of the TAG derivation tree, see Figure 3.1. Whereas such a change would require the rewriting of the existing derivation sequence when using another formalism such as CFGs. Putting it simply, TAGs exhibit a much smaller edit distance between structurally similar sentences than CFGs. Moreover, from a generative perspective for use with GE, TAG feasibility ensures that all intermediate steps during genotype-phenotype mapping are complete valid sentences that can be executed and evaluated.

The following subsections give a formal definition of TAGs, outline TAG derivation trees, and describe some interesting properties of the TAG formalism.

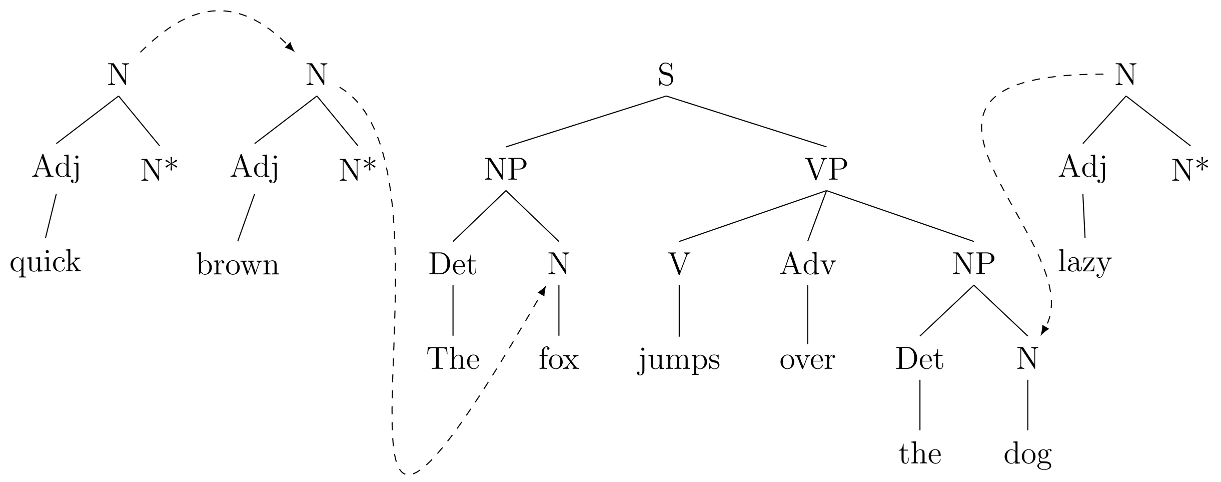


Figure 3.1: An example deformation of a TAG derived tree for the fox example. A dotted edge indicates an adjunction operation at the indicated node. The sub-trees at that node will be attached to the foot node of relevant auxiliary tree. Foot nodes are marked by *.

3.1.1 Definition of TAGs

This section provides a definition of TAGs, as well as a description of the composition operations used to construct TAG derivation trees.

Definition 10 (Tree-Adjoining Grammar) A TAG G is defined by a quintuple

$$G = (N, \Sigma, I, A, S)$$

where:

- N is a finite non-empty set of non-terminal symbols;
- Σ is a finite non-empty set of terminal symbols: $\Sigma \cap N = \emptyset$;
- S is the start symbol: $S \in N$;
- I is a finite non-empty set of finite trees. The trees in I are known as *initial trees* (or α trees). An initial tree has the following properties:

  - the root node of the tree is labelled with S;
  - the interior nodes are labelled with non-terminal symbols;
  - the leaf nodes, or the nodes along the frontier are labelled with terminal symbols;

  An initial tree represents a minimal non-recursive structured sentence or derivation tree produced by the grammar, i.e., it contains no recursive non-terminal symbols [114].

- A is a finite set of finite trees. The trees in A are called *auxiliary trees* (or β trees). An auxiliary tree has the following properties:

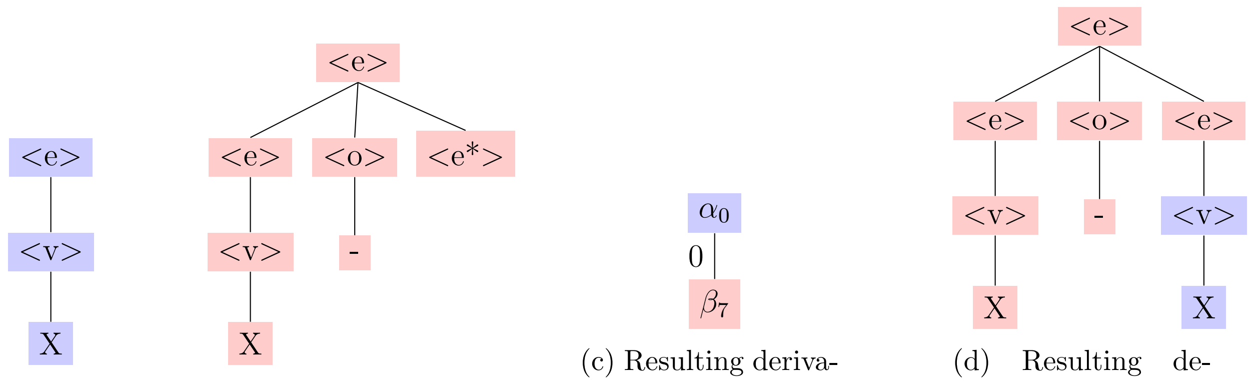
  - the interior nodes are labelled with non-terminal symbols;
  - the leaf nodes, or the nodes along the frontier are all labelled with terminal symbols apart from one node. This node, known as the foot node, is labelled with the same non-terminal symbol as the root node. The convention outlined in [99] is followed and foot nodes are marked with an asterisk (*).

  An auxiliary tree of type X represents a minimal recursive structure which allows recursion upon the non-terminal X during derivation [114].

The set of initial trees and the set of auxiliary trees together form the set of *elementary trees*, E, where $I \cap A = \emptyset$ and $I \cup A = E$. $\square$

Composition Operations

In TAGs, composition operations are used to modify existing structured sentences (trees), creating derived trees. In the original definition of TAGs [97], a single composition operation, adjunction, is used. Later, Schabes and Joshi [199, 98] include a second composition operation, substitution. The TAG formalism remains as powerful, regardless of whether



(a) α_0 (b) β_7 (c) Resulting derivation tree (d) Resulting derived tree

Figure 3.2: Composition operation: Adjunction. β_7 is adjoined to α_0 at α_0's root node, or address 0. The resulting derivation and derived trees are shown.

substitution is included or not, i.e., it produces the same set of strings and structures. However, the inclusion of substitution allows for a more compact formalism [99]. While this thesis makes exclusive use of the adjunction operation, see Section 5.1, both composition operations are described here for the purpose of completeness.

Adjunction The adjunction operation takes an initial or derived tree a, creating a new derived tree, d. This is accomplished by combining a with an auxiliary tree, b. A sub-tree, c, is selected from a. The type of the sub-tree (the symbol labelling the root of c), X, is used to select an auxiliary tree, b, of the same type. c is removed temporarily from a. b is then attached to a as a sub-tree in place of c. Finally, c is attached to b by replacing c's root with b's foot node. This operation is depicted in Figure 3.2.

Substitution In order to make use of the substitution operation, Definition 10 must be altered. The trees in I are no longer restricted to being the same type as S, and the trees in A no longer require all leaf nodes to be labelled with symbols from T. Leaf nodes in A may be labelled with symbols from N, if a tree exists in I of that type. These nodes are marked for substitution with a $\downarrow$. The new initial trees may then be substituted in place of non-terminal substitution leaf nodes, completing the tree.

3.1.2 TAG Derivation Trees

Unlike CFGs, TAG derivation trees and derived trees are distinctly different objects. The TAG *derivation tree* is an object tree, where each derivation node is labelled with an elementary tree, and each edge between derivation nodes declares the application of a composition operation. It is with the application of these composition operations that the TAG *derived tree* is constructed. Whereas the derivation tree is labelled with elementary trees, nodes in the derived tree are labelled with symbols, non-terminal symbols on the interior nodes, and terminal symbols along the frontier. The TAG derived tree is analogous with the CFG derivation tree.

While the general definition of a TAG derivation trees presented above is agreed upon, a number of variations exist. The derivation tree definition used by this thesis is that presented by Weir [214], and given in Hoai [78], whereby each of the edges between derivation nodes indicates an adjunction operation. Each node is labelled with a unique integer for that parent which represents a node index when traversing the elementary tree labelling the parent node. While Weir makes use of a pre-order traversal, this thesis utilises breadth-first traversal. Each integer is unique to that parent, i.e., adjunction may only be performed once at each *adjoinable* node. Therefore, a derivation node may have as many child nodes as there are adjoinable addresses in the elementary tree labelling that node. An adjoinable node is any node in an elementary tree which has not been adjoined to and is labelled with a symbol which is also the label of the root node of an auxiliary tree in A, the set of auxiliary trees. This property was coined the *non-fixed arity property* by Hoai [78]. As these derivation nodes are of a non-fixed arity, nodes may be added or removed without affecting the completeness of the derivation tree (and that of the derived tree). A basic example of a derivation tree is given in Figure 3.2, with further examples given in a later chapter in Figure 5.1a and Figure 5.2. The derived tree is generated from a derivation tree by applying each adjunction operation in post-order.

Variations on this definition exist, Schabes and Shieber [200] redefine the TAG derivation tree by allowing multiple adjunctions at the same adjoinable address, enabling trees of unbounded arity. Joshi and Schabes [98] modify the definition by Weir, enabling the use of substitution. However, the use of substitution in the derivation tree presents a problem, the non-fixed arity property is lost as elementary trees have a fixed number of nodes which require a substitution operation in order to consider the entire tree complete. When using substitution with TAGs the property of feasibility cannot be easily guaranteed as the composition operation are no longer operating on complete structured sentences. Hoai et al. [82, 78] address this problem by regarding substitution as an *in-node operation*, attaching a list of initial trees (called *lexemes*) to each derivation node. These lexemes are used for substitution at the nodes that are labelled for substitution (called *lexicons*) in the elementary trees labelling each derivation node. This approach retains the non-fixed arity and feasibility properties, while enabling a more compact formalism by making use of substitution.

3.1.3 Some Properties of TAGs

TAGs are more powerful than CFGs [96] which are commonly used in standard GE. The set of languages produced by TAGs, Tree-Adjoining Languages, is a super-set of Context-Free Languages (CFLs), those produced by CFGs [96]. It has been shown that for every CFG there is a TAG that is both weakly and strongly equivalent to it [96]. A grammar is weakly equivalent to another if it can produce the same language, or string set, as the other. Whereas a grammar is strongly equivalent only if it is both weakly equivalent, and can also represent each of the strings in the language using the same structures as the other, i.e., derivation trees. However, unlike CFGs, TAGs can also generate some context-sensitive languages [96, 99]. As such, there are also TAGs for which there are CFGs only capable of being weakly equivalent, as well as TAGs for which there are no weakly equivalent CFGs. While TAGs can generate more than CFLs, generating some context-sensitive languages, this extra power is limited and TAGs cannot generate all context-sensitive languages.

Despite this extra generative power being advantageous when using TAGs in the context of string generation and search, as is done with GBGP, other properties of TAG may also be helpful. The properties of feasibility and non-fixed arity ensure that both TAG derivation and derived trees are complete at every stage of derivation, having interesting repercussions for GBGP such as allowing evaluation of sentences at any stage of derivation (incremental evaluation) and the elimination of incomplete trees, and hence sentences, from population to name a couple. The following section outlines an algorithm for transforming existing CFGs into TAGs.

3.1.4 CFG-TAG Transformation

A lexicalised TAG (LTAG) is a special instance of a TAG. A lexicalised grammar has two defining properties:

- the grammar consists of a finite set of structures, each structure with at least one terminal symbol, known as the anchor;
- it has at least one operation for composing these structures together.

The TAGs referenced to in the continuation of this thesis are LTAGs, as all leaf nodes of the elementary trees, with the exception of foot nodes, are labelled with terminal symbols. The terms TAG and LTAG will be synonymous throughout and will both refer to lexicalised TAGs.

Joshi and Schabes [99] state that for a *"finitely ambiguous CFG*[1] *which does not generate the empty string, there is a lexicalised TAG generating the same language and tree set as that CFG"*. Joshi and Schabes also provided an algorithm for generating such a TAG. This algorithm, presented in Algorithm 3.1, as well as being outlined below, allows existing CFGs used by GE to be transformed into TAGs. It also enables TAGs to be created by designing new CFGs. This is beneficial as CFGs can be more trivially designed by hand. Figure 3.3 provides an example of a TAG produced by this algorithm.

The algorithm proceeds as follows. Given a finitely ambiguous CFG,

$$G = \{N, \Sigma, P, S\}$$

where N is the set of non-terminal symbols, Σ is the set of terminal symbols, P is the set of production rules, and S is the start symbol:

1. Construct a directed graph, g, from G. The nodes of g are labelled with symbols from N and the edges of g are labelled with the productions from P which map between them;

2. Find the set of minimal cycles, c, in g such that they contain no other cycles within them;

3. The productions in P are then divided into two separate sets, R is the set of recursive productions, and NR is the set of non-recursive productions in the grammar. A production is recursive if it is part of a cycle, c_i;

4. Using S as the root node, create the set of all possible derivation trees using only the productions in NR. This is the set of initial trees, I;

[1]A grammar is said to be *finitely ambiguous* if all finite length sentences produced by that grammar cannot be analysed in an infinite number of ways.

5. Create A, the set of auxiliary tree, as an empty set. $A = \emptyset$;

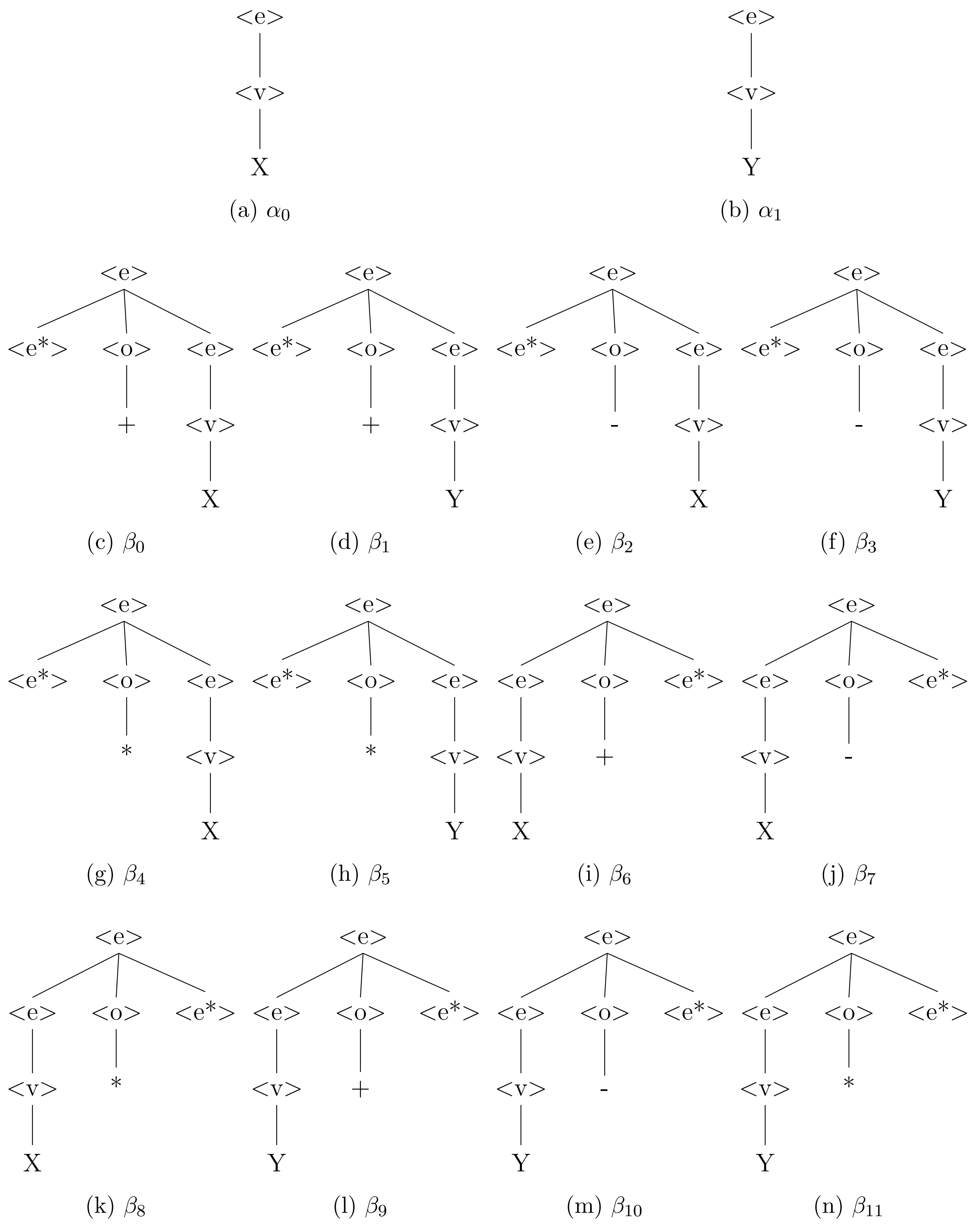
6. For each node n_j, in each of the cycles c_i, if there is a tree in $I \cup A$ that contains a node which has the same label as n_j, then create the set of all possible derivation trees using only the productions in NR and the current cycle where the n_j is the root node and the foot node is the node with the same label as n_j. Add this set of trees to A;

7. This continues until all cycles have been processed.

The resulting TAG is both weakly and strongly equivalent to the source CFGs. That is to say, the TAG can produce the same language and the same tree set as the source CFG. Algorithm 3.1 provides a concise summary of this algorithm, with an example of a TAG produced by the algorithm can be seen in Figure 3.3. This TAG was generated from the CFG shown in Figure 2.2. The following section outlines how TAGs have been applied to date in the field of EC, with a particular focus on the field of GBGP.

Algorithm 3.1 Generating a TAG from a CFG.

```
Require: $G = \{N, T, P, S\}$ {A CFG}
  $g \leftarrow$ createDiGraph($G$)
  $c \leftarrow$ findBaseCycles($g$)
  $R \leftarrow$ getRecursiveProductions($P$, $g$)
  $NR \leftarrow P - R$;
  $I \leftarrow$ generateTreesByExpansionEnumeration($S$, $NR$)
  $A \leftarrow \emptyset$
  for all $c_i$ in $c$ do
    for all $n_j$ in $c_i$ do
      $E \leftarrow I \cup A$
      if a tree in $E$ has a node labelled the same as $n_j$ then
        $A \leftarrow A \cup$ generateTreesByExpansionEnumeration($n_j$, $c_i$, $NR$)
```



(a) α_0 (b) α_1

(c) β_0 (d) β_1 (e) β_2 (f) β_3

(g) β_4 (h) β_5 (i) β_6 (j) β_7

(k) β_8 (l) β_9 (m) β_{10} (n) β_{11}

Figure 3.3: The set of initial trees, $I = \{\alpha_0, \alpha_1\}$, and the set of auxiliary trees, $A = \{\beta_0, \beta_1, \ldots, \beta_{11}\}$, generated by the transformation of the CFG shown in Figure 2.2 into a TAG using Algorithm 3.1.

3.2 Related Research

TAGs, which have their basis in the fields of NLP and linguistics, have also been made use of in the field of EC. Much work has been done by Hoai [78], amongst others, in incorporating TAGs with GP. A number of different frameworks using TAG based representations for GP having been proposed [86, 82, 91]. This section gives an overview of these frameworks and those studies, amongst others, which have been performed using each of the frameworks as the underlying method.

3.2.1 Linear TAG

Originally, Tree Adjoining Grammar Guided Genetic Programming (TAG3P) [86], a framework for grammar guided genetic programming using TAGs was introduced. TAG3P makes use of a linear genotype, beginning with an initial tree, followed by a series of auxiliary tree - adjoinable address pairs. No substitution operation is used with TAG3P. The adjunction operations defined in the genotype are applied from right to left, producing a TAG derived tree with the phenotype along the frontier. Operators, which are applied to the linear genotype, are restricted to ensure legal genotypes are produced.

TAG3P has been applied to many problems including symbolic regression problems of increasing complexity [75, 77, 147], as well as being used to discover alternate representations of trigonometric identities [76, 149]. In both cases TAG3P is compared against standard GP [113] and CFG-GP by Whigham [215] (CFG-GP is referred to as GGGP by Hoai [75]). For the symbolic regression problems, it is shown that TAG3P performs better, in particular on problems with target functions of increasing structural complexity. Whereas when being used to find trigonometric identities, TAG3P appears to find and make use of useful building blocks or groups of auxiliary trees while constructing approximate solutions to the identities. Further work is successfully completed in an attempt to evolve exact solutions to these identities by making use of local adjunction constraints, i.e., selective, obligatory and null adjunction [114], in order to bias the search. Hoai et al. [81] provides a summary of these results.

In addition, Nguyen et al. [148] goes on to investigate how TAG3P performs on the Boolean even-n parity class of problems. The motivation being that the use of building blocks by TAG3P seen previously would be advantageous. TAG3P is compared against GP and CFG-GP on even three, four and five parity problems. In this case, TAG3P performs worse on all three problems, failing to solve even four and even five. Nguyen et al. posit that the search space is not suitable for the promotion of building blocks and that as the size of the problem increases, much larger solutions are required which TAG3P is not able to create.

More recently, Kim et al. [107] has investigated operator self-adaptation in the context of TAG3P. Comparing a number of different methods for setting operator application rates and rewards policies for those approaches, it is shown that TAG3P benefits from operator rate adaptation.

3.2.2 Ant Colony Optimisation TAG

Subsequent to the introduction of TAG3P, the approach is extended in the form of AntTAG [1]. AntTAG, an ant colony optimisation algorithm, makes use a modified form of the TAG3P representation. Whereas TAG3P uses a population based approach of linear TAG genotypes, AntTAG utilises a *pheromone table* representation which is used to generate linear TAG genotypes. A pheromone table is a transition table, which lists the probabilities of any one elementary tree being adjoined to another at a specific addresses. A solution is constructed by iteratively transitioning through this table, defining a sequence of adjunction operations from one elementary tree to another. This continues until a special termination symbol is found, at which point a linear TAG genome has been constructed and evaluation can proceed as in TAG3P. Single point crossover is used on the produced linear genotypes and if fitter individuals are created then the pheromone table is updated to reflect this. Due to TAG feasibility, valid individuals are always produced.

AntTAG is initially benchmarked on the quartic polynomial symbolic regression problem, with much success. Following this, work by Shan et al. [202] expands upon the approach, identifying potential reasons for AntTAG's previous success and redesigning the pheromone update rules to improve performance. In addition, a further modified pheromone update rule is applied to two more complicated symbolic regression problems. AntTAG also proves successful on these problems, outperforming TAG3P.

3.2.3 Tree-based TAG

The TAG3P framework is later modified in the form of TAG3P+ [82]. Rather than the linear genotype used by TAG3P, this new representation makes use of a TAG derivation tree based genotype, which supports substitution as an in-node operation, see Section 3.1.2. The operations used are tree based, including sub-tree crossover and mutation. The inclusion of substitution allows for a much more compact representation and the non-fixed arity of the derivation nodes in the genotype open up many opportunities for new types of operators. Initial studies investigate the effect of bias on the six multiplexer problem. Preferential, structural and lexical bias are implemented by changing adjunction and substitution probabilities (user defined). Initial unsuccessful results help discover that subtree crossover limits the effect of the imposed biases, and that by modifying crossover to preserve these biases and the distribution of lexemes and auxiliary trees, a statistically significant improvement is observed.

This framework is later adopted by Wang et al. [212] for story plot generation. Story skeletons, uninstantiated plots, are encoded in TAG derivation trees and evolved using an interactive fitness function.

Initialisation

Following this, Hao et al. [61] makes use of a component of the TAG3P+ framework to highlight the importance of the structure of initial populations in CFG-GP. Using CFG-GP [215], two methods of initialisation are compared, the TAG3P+ initialisation method, which allows for a uniform distribution of initial tree sizes, transforming the initial derivation trees into derived trees which are equivalent to CFG-GP derivation trees, and the ramped random initialisation method used by CFG-GP. Hao et al. [61] found that populations initialised using the TAG3P+ method far outperformed those created using the CFG-GP method.

Hoai et al. [83] continue with this examination of initialisation. TAG3P+ is compared to GP with both representations making use of populations initialised by the TAG3P+ initialisation method, as well as comparing against GP initialised by the popular ramped half and half method [113]. The study makes use of the evolutionary multi-objective optimisation algorithm SPEA2 [223], optimising for fitness and phenotype size on two problems, the quartic symbolic regression and the six multiplexer problems. It is found that TAG3P+ outperforms GP with both commonly and independently initialised populations. TAG3P+ populations are observed to converge faster towards optimally size individuals, resulting in more search occurring around the optimal size rather than in space of larger individuals like GP. Hoai et al. [83] state that, while the TAG representation used gives some advantages, bounds on search spaces in the previous studies contribute largely towards TAG3P+'s improved performance when compared with GP.

Operators

As a result of the non-fixed arity property of the derivation tree representation used, a number of different operators have been explored which affect the search in different ways. The effects of three operators are investigated: node insertion and its reciprocal node deletion [80], sub-tree relocation [150], and sub-tree duplication with its reciprocal operator, truncation [84]. All three operator sets are examined as local search operators for use with a hybridised evolutionary and local search algorithm, whereas node insertion and deletion, and duplication and truncation are also examined for use as mutation operators in the TAG3P+ framework.

The insertion and deletion operators, which insert or remove nodes from the frontier of derivation trees, are compared as mutation operators for performance on the six multiplexer, quartic symbolic regression and digital circuit synthesis problems [80]. They are also compared as local search operators, with a number of local search steps occurring between generations. Population sizes are scaled to ensure that the total number of fitness evaluations are identical on all setups. It is found that these operators perform best when used as mutation operators in conjunction with sub-tree crossover. It is also found that local search operators perform well on the multiplexer problem, and in general, with smaller populations, the use of the operators for local search is beneficial. This study also shows that local search alone is not sufficient to solve the problems.

Nguyen et al. [150] further extends TAG3P+, making use of relocation as a local search operator. Relocation is tree-based operator biologically inspired by conservative genetic transposition. Relocation swaps a sub-tree with a NULL node, moving a sub-tree of adjoined auxiliary trees to a new location in the derivation tree. NULL nodes are attached to all non-adjuncted adjoinable addresses in the derivation tree. This operation is easily accomplished due to the non-fixed arity representation. The effectiveness of the operator for local search is tested on two similar symbolic regression problems of increasing complexity, a difference in volume between two cuboids problem and the trigonometric identities problem. The new local search operator joined with sub-tree crossover is compared against GP and TAG3P+ with sub-tree crossover and mutation. Again, when comparing local search to non-local search the population size is scaled as a function of the number of local search events, maintaining an equal number of fitness evaluations. GP and TAG3P+ are also compared with similarly scaled population sizes but more with increased generations. The results find that relocation as a local search operator outperforms GP and TAG3P+ on all problems. The final set of experiments show that the reduction in population size does not contribute to the increase in performance.

An additional biologically inspired operator, replicative genetic transposition, or duplication, is also proposed [84]. Duplication takes an existing derivation sub-tree of adjoined auxiliary trees and replaces a NULL node in the frontier of the derivation tree with a copy of that sub-tree. It is found that the sole use of duplication results in extreme growth, therefore, a companion operator, truncation, is included which replaces an existing sub-tree with a NULL node. These operators are, again, compared with GP and standard TAG3P+, as both mutation operators and further as local search operators, with scaled population sizes, on a series of six symbolic regression problems of increasing complexity. As mutation operators, it is found that duplication and truncation perform better on three of the six instances, however, the improvement is not statistically significant when compared to TAG3P+ using just sub-tree crossover. As local search operators, the combination of the two operators, in conjunction with sub-tree crossover and mutation, outperforms all other setups, maintaining reliable performance with small population sizes even when the problem complexity increases. Further tests are performed with GP and standard TAG3P+ ruling out the small population sizes as a contributor to the boost in performance.

While each of these operators is examined on a number of problems independently, to ascertain whether their effects overlap and to better understand their function, Kim et al. [108] provide a further detailed comparison of these operators and more. In this study it is shown that each of these operators have independent useful effects when used on certain problem types. Following this, Kim et al. [109] expand upon their previous work and apply operator rate self-adaptation to this diverse selection of TAG3P+ operators, comparing against a number of self-adaptation techniques on a real-coded GA and TAG3P+.

Structural Difficulty

Daida et al. [39] identify difficulties with the structural search spaces in GP, attributing this problem to the tree based representation commonly used by GP. Hoai and McKay [79, 85] propose that these difficulties might be a result of the fixed arity property of the tree based GP representation. By transforming fixed arity GP trees into non-fixed arity TAG derivation trees, and applying a TAG-based hill climbing algorithm, utilising node insertion and deletion as local search operators, Hoai et al. [85] show that the effect of this problem, as seen when solving the LID problem [39], can be largely removed.

3.2.4 Incremental and Developmental Evaluation

McKay et al. have argued that if an individual is evaluated multiple times throughout its development, then modular structure, inherent to the representation, can provide an adaptive advantage to that individual. In addressing this argument, McKay et al. [135, 89] extend TAG3P+ to make use of incremental evaluation, in the form of DEVTAG. Evaluation in DEVTAG is analogous to the development of biological organisms, evaluating an individual at different stages of development. This is made possible by the feasibility property of TAG derivation trees. A number of depth limits are set, whereby derivation of an individual is ignored below each limit. This results in a number of different phenotypes of increasing size and complexity for a single genotype.

During selection, individuals are compared a number of times with derivation stopping at each successive depth limit. At each depth limit, a less complex version of the full problem being solved is presented to all individuals, this continues, deriving the individuals to the next depth limit, until one individual obtains a greater fitness value on any one of the incrementally more complex problems.

DEVTAG [135] is initially applied to the symbolic regression problem $\sum_{i=1}^{n} x^i$ for all versions of the problem from $n = 1$ to $n = 9$, testing individuals successively on incrementing values of n. It is shown that DEVTAG outperforms TAG3P+ and GP on solving the final form of the problem where $n = 9$. DEVTAG also results in fewer total fitness evaluations and smaller final solutions. Further analysis is also performed on a Fourier series symbolic regression problem [89], showing that the success of DEVTAG is partially related to the TAG representation and the incremental process. However, the entire developmental process, including sub-problems of increasing complexity, is required to see the full advantage.

Hoang et al.[91, 88] go on to extend this system, improving upon DEVTAG. Two problems observed with DEVTAG is that the size of each developmental stage is user defined, which is not ideal when applying to unfamiliar problems. In addition, there is limited modularity in the representation, with the Fourier series problem requiring two auxiliary trees sharing common sub-trees as a module to solve the problem efficiently. DTAG3P addresses both problems, a novel developmental approach to GP using TAG based L-systems. The representation consists of L-system expansion rules, with each rule mapping from an L-system predecessor to successors. Successors in this case are TAG derivation sub-trees, containing both elementary trees and predecessors. During selection, similar to DEVTAG, evaluation is incremental. However, rather than using a depth limit, L-system expansions are used. At first, the predecessors in an individual's initial derivation tree are ignored, evaluating the individual on the most basic sub-problem, comparing that individual to others, with only the best individuals being retained. If more than one individual remains, this continues, with the predecessors in those trees being replaced by their successors, creating larger more complex trees. These new derivation trees are evaluated on the next instance of the problem. This repeats until only one individual remains or the final instance of the problem is reached. In this case random selection is used. This approach of evolving the expansion rules allows for the generation of useful modules, as well as allowing evolution to determine the size of each developmental stage. While this removes some parameters, it also introduces some more, including the number of L-system rules, and the minimum and maximum size for initialising these rules. It is shown that DTAG3P performs better and finds solutions much quicker than DEVTAG. An overview of these systems is given by Hoang et al. [90]. In addition, Hoang et al. [92] goes on to show that the DTAG3P system generates general, rather than specialised, solutions.

The inspiration for these systems is that regularity in the genome is promoted by evolution when using developmental evaluation, and that this increase in regularity correlates with fit individuals. A series of studies examine this claim [100, 206, 136, 132]. Kang et al. [100] apply tree compression algorithms to genomes of GP, TAG3P+, DEVTAG and DTAG3P. Their hypothesis being, that genomes with increased levels of regularity are more amenable to being compressed. Equivalent Decision Simplification is used to simplify the trees, ensuring redundant code, or bloat, is not being compressed and XMLPPM is used to compress the trees. DTAG3P trees are found to compress well, whereas DEVTAG, TAG3P and GP trees do not. In addition, problems with the compression metric due to small trees can cause issues. Shin et al. [206] follow this with further analysis, using an improved compression metric, the duplication and truncation operator in TAG3P are examined, as well as further examination of DTAG3P and DEVTAG. A comprehensive analysis of the culmination of the work in both of these studies is provided by McKay et al. [136]. While these studies give static estimates of the regularity within the genotypes of each generation independently, no information about the propagation of individual building blocks throughout evolution is available. McKay et al. [132] extend these studies to estimate the extent of which frequent building blocks from one generation propagate to the next. A comprehensive overview of the developmental evaluation approaches presented in this section is given by Hoang et al. [93].

While the literature on TAGs in EC presented in this section is numerous, the bulk of the work is on tree-based GBGP representations.

3.3 Summary

This chapter provides an introduction into TAGs, the grammar type with which this thesis examines. A formal description of TAGs is given, as well describing a number of properties of the TAG formalism which are needed to develop a TAG-based representation for GE. Following this, a comprehensive literature review is provided on the use of TAGs in the EC field, in particular for use with GP.

The following chapter is the final background chapter, and provides information into the use of developmental systems, gene regulatory network in particular, in the field of EC to date.

Chapter 4

Developmental Evolutionary Algorithms

The view of classical genetics, that genes directly control every feature, structure and cell type within an organism, have been a very successful source of inspiration in the field of EC. However, this view has been largely replaced in modern genetics. It has been discovered that genes can be responsible for many different effects and, in fact, there are great networks of interacting genes, molecules and other physical systems which work together to produce the different cells, tissues, organs and fully functional organisms we see in nature. These complex networks of interaction are still a subject of great study today.

As a results of this, the field of Evolutionary Computation has had a recent increase of interest in developmental biology, investigating different approaches of emulating this complex developmental mapping from genotype to phenotype. To date, much of the work that has been done with evolutionary developmental systems in the field of GP specifically has been at a high-level and ontogenetic in nature. That is to say, it has been concerned with the growth or morphogenesis of individuals by means of interaction with the environment [207, 91, 62]. These developmental systems derive the phenotype from the genotype, with the effect of the phenotypes being twofold. First being evaluated on the problem being solved, the phenotype then undergoes some morphogenesis defined within itself. This process continues, traditionally creating larger and more complex phenotypes which can survive in problem environments of greater complexity. The morphologies on which these systems are modelled on, however, are themselves the result of lower-level regulatory processes, at least part of which is genetic and is known as gene regulatory networks (GRN). This thesis is interested in the exploration of the combination of these GRNs and the properties of TAGs.

This chapter is split into two main sections. The first, Section 4.1, provides a survey on the use of GRNs within the field of artificial evolution and EC. Following this, Section 4.2 gives a detailed description of a particular model of artificial GRNs which has been designed with EC in mind, the Banzhaf artificial GRN model, and its properties and extensions. This GRN model will be further discussed in Chapter 9.

4.1 Related Research

Early work by Eggenberger [45] presents a system based on differential gene expression to control the developmental processes of three-dimensional multi-cellular organisms. Integer genomes representing GRNs of regulating and "structural" genes are evolved using a simple GA. Genes in simple cells on a three-dimensional grid can self regulate, as well as communicate with neighbouring cells, influencing their regulation. Eggenberger uses the placement of morphogen sources in the grid, and the gradual decrease of morphogen concentration further from the source, to drive the development of these organisms.

Separately, Reil [192] proposes an artificial genome model with biologically plausible properties. Properties such as promoter regions to define genes in the genome and regulatory sequences which interact with gene products to regulate gene activity. Reil defines gene activity as a binary process, with the presence of a single enhancer being sufficient to activate a gene. However, if a gene is also inhibited then that gene will not activate. As a result of this binary activation model, gene activation can be visualised on an expression graph from which behaviours which mirror real gene expression can be seen. The behaviours include cyclic gene activity, the ability to differentiate into different attractors due to external signals, as well as showing a high degree of robustness.

More recently, Willadsen and Wiles [217] examine this artificial genome model in relation with an earlier Random Boolean Node (RBN) model [101]. Chaotic attractors are prevalent in the highly connected RBNs. Willadsen and Wiles investigate the relationship between network connectivity and inhibition in the artificial genome model. It is shown that the interaction of these two properties in the space of artificial genomes drives the occurrence of chaotic attractors, with higher levels of connectivity and medium levels of inhibition maximising their occurrence.

Further to this, Hallinan and Wiles [60] highlight the biologically implausible properties which are seen in existing GRN models, in particular the synchronous node update model. They state that biological cells are assumed to function on the "edge of chaos", between totally frozen and chaotic dynamics, a region characterised by limit cycle attractors which are widely assumed to be models of cell types [101]. Hallinan and Wiles extend Reil's model to investigate the relationship between synchronous and asynchronous updating, and the occurrence of limit cycles. It is shown that the dynamic behaviour of the artificial genome model collapses to a single point attractor in almost all cases when using the more biologically plausible asynchronous updating.

Watson et al. [213] extend this model integrating it with a developmental phenotype in the form of L-systems. Genes in the artificial genome model are assigned to classes. Specific classes of genes are mapped to parameters of the L-system phenotypes, with the number of regulated genes of a certain class relating to the value of a parameter. Watson et al. examine the effect of genome mutation on the resulting phenotypes. This study highlights the adaptability of the artificial genome and is a novel application.

Separate to this, Banzhaf [6] considers the idea that successful evolutionary design is best achieved with a networked system, and that emergent phenomena in nature require networks to become permanent. Banzhaf [5] goes on outline a new model of GRNs for use as representation for GP. This model extends on previous work by including real-valued gene activation and inhibition as well as using a fully connected topology, with the strength of each edge limited by the matching of gene products (proteins) and gene regulatory sites. The model is shown to generate interesting dynamics including heterochrony, and to be capable of capturing essential features of natural GRNs [4, 118].

A number of studies have been performed into examining the different network motifs [7] and network topologies that the model is capable of generating. These include small world and scale-free topologies [119, 154]. Scale-free network topologies, as are seen in natural GRNs, are desirable as these networks are highly tolerant to failure.

In order to use make use of this model for evolutionary computation modifications are required. Nicolau et al. [156] extend the model to be used as a computational device. Two extra sets of proteins are used: extra transcription factor proteins, independent of the genome, are added to be used as input; and a second promoter sequence is used to detect a second class of gene along the genome, product, or P-genes, produce non-regulating proteins which are used as output. This approach has been shown to be successful at solving a number of different problems [156, 158].

Separately, Lopes and Costa [121] extend the Banzhaf model in the form of ReNCoDe. Using the strength of inhibition and activation along each edge within the network, the GRN graph structure is transformed into an executable program-graph. Functions and terminals are mapped to the nodes by means of a majority rule of the 32 bit protein signature of each node. This approach is shown to perform very well on a number of benchmark problems [121]. This method is extended to allow recurrent connections between nodes, enabling generation of general, or incremental solutions. This extension is shown to perform well on the n-Fibonacci and Even-n problems, as well as the sum of squares problem [122] which highlights an advantage of the recurrent connection extension, solving the problem using linear operations. An overview of this work is presented by Lopes and Costa [123].

More recent work presents a combination of ReNCoDe with the Nicolau extension [124]. This extension makes use of a P-gene as the output node of the ReNCode program-graph. If multiple P-genes exist in the graph, then each program-graph with a different P-gene node as the root is tested. Additionally, ReNCoDe is combined with GE [125]. This extension allows the GRN to iterate a number of times before sorting the proteins by concentration, and dividing the protein signatures into 8 bit regions to be used as integer codon values for mapping. The approach is shown to achieve a high degree of success on the artificial ant problem, however, the study highlights the fact that many evaluations are required. When the number of evaluations is decreased, by increasing the number of GRN iterations, performance decreases.

Another popular model of GRNs used in EC is that of fractal proteins by Bentley [11]. This novel EC system makes use of a "fractal chemistry" made up of interactions between square Mandelbrot subsets. This model uses a genome of real values, with genes defining a length three promoter sequence, a length three coding region and a threshold value used for activation. Each length three region represents a square subset of the Mandelbrot set, the first two components specifying the center coordinate and the third specifying the length of side. Protein matching is performed by overlapping the promoter region subset with the subsets of the proteins in the system and activating if a certain threshold of matching, or overlapping is achieved. The system also supports multiple cells interacting. A number of different studies extending this system have been published [221, 220, 117, 115, 116].

Other notable works on GRNs in EC include: the study of the evolvability, or the capacity to facilitate effective search, in relation to direct vs indirect encodings by Reisinger et al. [193]; the presentation of an artificial ontogeny system by Bongard [13] which grows virtual agents from GRNs, investigates their emergent behaviours and evaluates them in a

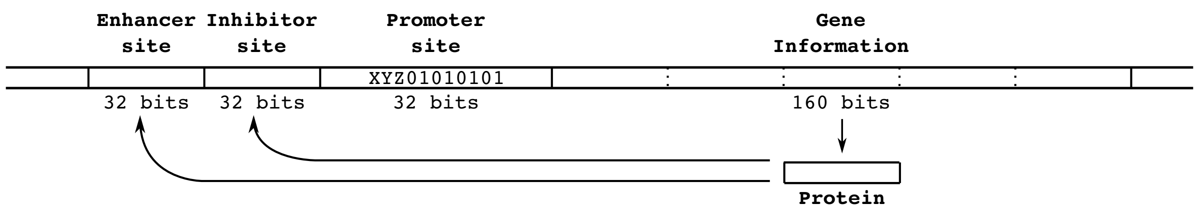


Figure 4.1: A simplified view of a gene on the genome split into its different sections.

physics-based three-dimensional environment; and the multi-layered artificial embryogeny system by Zhan et al. [222]. The Banzhaf model and the Nicolau extension are described in detail in the following section.

4.2 Banzhaf Artificial GRN Model

The Banzhaf GRN model [5] consists of three components: a genome, genes, and proteins. The model mimics, at a greatly simplified level, the biological interaction of proteins with the genes of a cell. By binding at *regulatory sites*, certain proteins can regulate the expression of genes, and hence, the production of additional proteins. Proteins are assigned concentration levels in the model, with a total concentration of 1.0 for each type of protein. The term *concentration*, in the context of this model, represents the proportion, or amount, of a particular protein to the rest of the proteins in the cell. Concentration is a normalised quantity such that the concentrations of all proteins sum to 1.0.

In order to properly describe this model, a number of definitions are required. Each of the components of the model shown in Figure 4.1 are defined below.

Definition 11 (Chromosome) A chromosome, or genome, is a finite length binary string. In this case, a GE chromosome is used. □

Integer strings can also be used with the assumption that an integer is represented by 32 bits.

Definition 12 (Protein) A protein consists of a 32 bit signature, separate from the chromosome, and a concentration level between ZERO and 1.0. Proteins are the active component of the model, with their concentration levels being regulated by their interaction with other components of the model. □

ZERO is usually some positive number close to zero. If concentration levels were to reach zero, that protein would not be able to re-emerge as existing concentration levels are used when calculating expression rates. A value of 1e-10 is used by this study.

Definition 13 (Regulatory Site) A regulatory site is a 32 bit region along the chromosome which interacts with proteins. This interaction results in regulatory signals. The model has two types of regulatory sites, *enhancer* and *inhibitor* sites. These sites are found next to each other and affect regulation positively or negatively respectively. Positive regulation of a gene increases the production rate of that gene's protein, whereas negative regulation decreases it. □

Definition 14 (Promoter Site) Promoter sites are 32 bit regions along the chromosome. They are of the form `XYZ01010101`, where `XYZ` is an arbitrary 24 bit sequence with only the final eight bits being of significance. A specific sequence of eight bits identifies the location of a *gene* along the chromosome. □

Definition 15 (Gene) A gene is a region of 256 bits along the genome. The function of a gene is to produce a protein. The rate at which this protein is produced depends upon the interaction of the proteins in the system with the gene's regulatory sites.

The location of a gene is identified by the specific eight bit promoter sequence along the genome. Directly to the left of the promoter site are two regulatory sites, an inhibitor site, followed by an enhancer site. While to the right of the promoter site is a 160 bit region which encodes the protein. This region is split into five 32 bit sections which are used to decode the protein produced by this gene.

1 **1** 1 0 **1 0** 0 1 0 1 **1 0 1 0 0** 0 **0 1 0 0 0 0** 1 0 **0 1 1 1** 0 **0** 1
0 1 0 1 0 **0 1 0 1 0** 0 **0 1** 1 1 **1 0 1 0** 1 **0** 1 **0 1** 1 **1 1** 0 **1 0** 1
0 1 1 **1 1** 1 **1** 1 0 1 **1 0 1 0** 1 0 **0 1 0 0** 1 **0** 1 0 **0** 0 **1** 0 **1 0 0**
0 1 0 1 0 1 **1 0 1 0 1 0** 0 1 **0 1** 1 **1 0** 1 **0 0 0 1 0 1** 0 **1 1 0 0**
1 0 **0** 0 **1 0 1 0 1 0 1 0 1 0 0 1 0 1** 1 **0** 1 **0 0 1 0 1** 0 **1** 0 **0 0**

---

0 1 0 1 1 0 1 0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 1 1 1 0 0

Figure 4.2: The protein signature is calculated by performing a majority rule vote for each bit across the five 32 bit regions at the end of the gene. For each bit, read the corresponding bits from each of the five 32 bit regions. If there are more than two set bits, set the protein bit. Otherwise, leave the bit unset.

The production rate of the protein encoded by these 160 bits is regulated by the interaction of the regulatory sites on the same gene with all of the proteins in the system. □

4.2.1 Protein Decoding

In order to express a gene, i.e., to decode and produce a protein, the 32 bit signature of that protein must be calculated. The five sections making up the 160 bit region at the end of the gene are taken and a majority vote is performed on each of the respective bits across the five sections. For example, to decode the first bit of a protein's 32 bit signature, a majority vote is performed using the first bit of each of the five sections. If more than two of these bits are set, then the first bit of the signature is set, or given the value of one. Otherwise the first bit remains unset, or a value of zero. Figure. 4.2 provides an example of this process.

4.2.2 Protein Regulation

Proteins regulate the expression of genes, and hence the production of additional protein. The effect that a protein has on the expression of a gene is calculated by examining the degree of matching between that protein and the two regulatory sites of the gene (enhancer and inhibitor) in combination with the current concentration of that protein. This is calculated by taking the XOR of the protein signature with that of each regulatory site. The number of bits set in the resulting 32 bit sequence is the degree of match between the two, i.e., the number of complementary bits between them. The enhancing, e_{it}, and inhibiting, h_{it}, signals for the expression of gene g_i at time t are calculated as follows:

$$e_{it}, h_{it} = \frac{1}{N}\sum_{j=1}^{N} c_{jt} \mathrm{e}^{\beta(u_j - u_{max})}$$

where N is the total number of proteins, c_{jt} is the concentration of protein j at time t, u_j is the number of complementary bits between the regulatory site and protein j, u_{max} is the maximum number of complementary bits observed in the system so far, and β is a positive scaling factor. A value of 1.0 is used for scaling factors β and δ across all experiments.

The expression rate of gene g_i (the production of p_i) at time $t+1$ is given as:

$$\frac{dc_{it}}{dt} = \delta(e_{it} - h_{it})c_{it}$$

where δ is a scaling factor. The new concentration values for each protein, $c_{i(t+1)}$ are then calculated and finally normalised to sum to 1.0.

$$\begin{aligned} c_{it} &= c_{it} + \frac{dc_{it}}{dt} \\ c_{i(t+1)} &= \frac{c_{it}}{\sum_{j=1}^{N} c_{jt}} \end{aligned}$$

4.2.3 Operation

To construct a GRN, the chromosome is scanned for matching promoter sequences and the genes are extracted from those locations. Proteins can then be decoded from the genes. Each gene encodes a single protein.

The network is constructed from these genes and proteins, and can be represented as a *fully connected directed graph*, with each gene as a node in that graph. The directed edges in the graph represent protein interactions, from the gene coding each protein to all other genes. Each edge can be labelled with the strength of the regulatory signal of the decoded protein coupled with the gene. The sum of these signals, positive (enhancing) and negative (inhibiting), can be considered analogous to weights in recurrent neural networks.

In its initial state, the proteins of the GRN are given a starting concentration value. Typically this value is $\frac{1}{||P||}$, where $||P||$ is the number of proteins in the model. At each iteration of the model, protein regulation occurs *synchronously*, as described in Section 4.2.2. The concentration levels of each protein are observed at each iteration, and it is through the interaction of these proteins with the genes in the system that produces the complex non-linear dynamics. Examples of dynamics produced by the model are given in Figure 4.3.

4.2.4 I/O Extension

While the GRN model shows a good ability to produce complex dynamics, the model is limited in use, in particular, as a computational tool. Nicolau et al. [156] address this problem by extending the model with input and output capabilities such that it might be used computationally. The ability to effect change in, and get feedback from the system is incorporated by adding input and output proteins. This extension allows the GRN model to interact with dynamic environments, both by reacting to, and affecting that environment by means of its input/output capabilities. This extended model has been successfully applied to a number of problems [156, 158].

I/O

Genes are split into two groups in this model, *Transcription Factors* (TF-genes) and *Products* (P-genes). Each gene type produces a corresponding type of protein, *TF-proteins* and

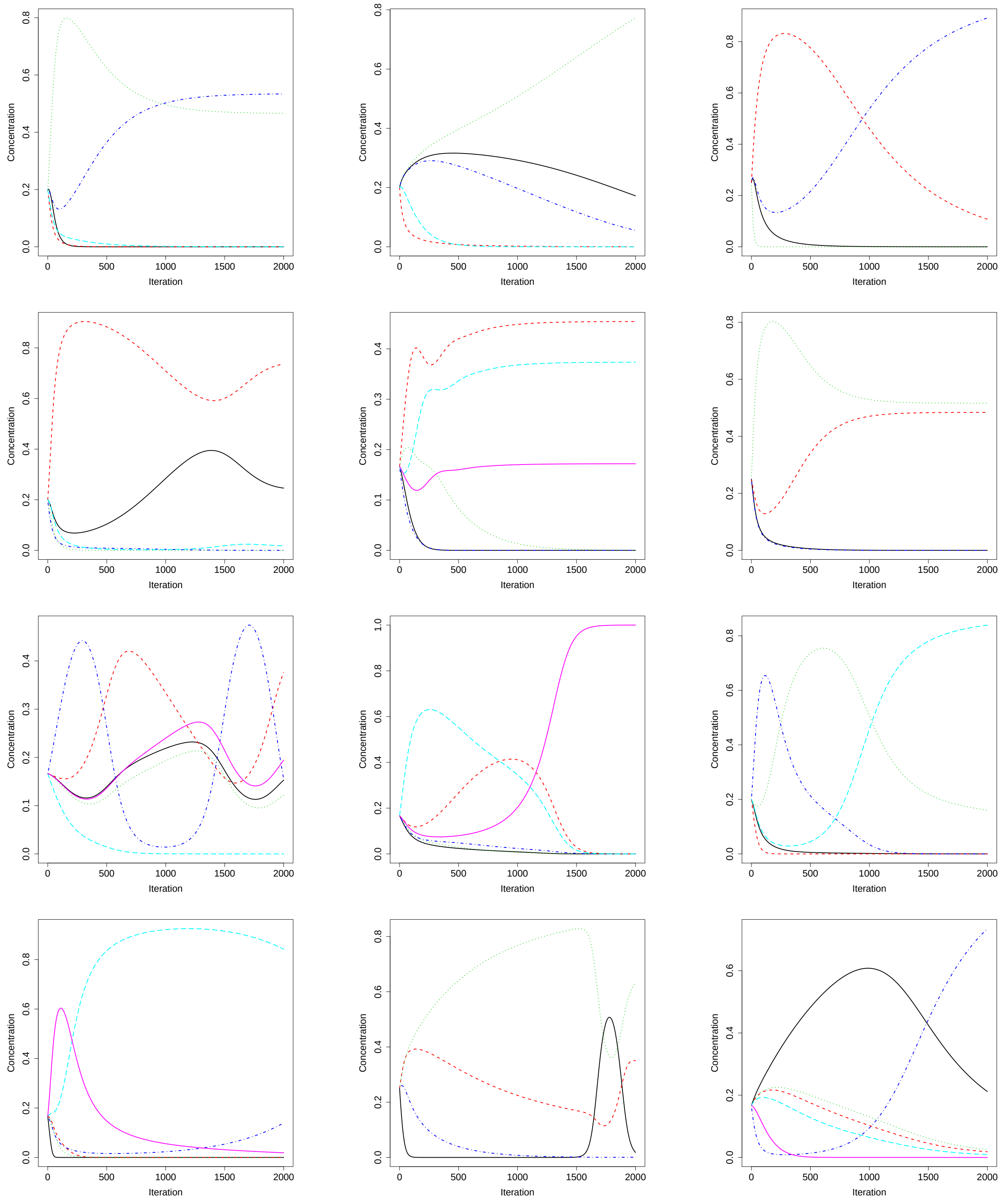


Figure 4.3: A sample of example protein concentration dynamics generated using the Banzhaf GRN model implemented in DTAGE.

P-proteins. A gene's type is determined by its promoter signature, e.g., `XYZ00000000` can be used to identify TF-genes and `XYZ11111111` to identify P-genes. TF-genes produce TF-proteins which can bind to regulatory sites and affect gene expression. P-genes produce P-proteins which are used solely as output and, as such, are prevented from binding and affecting regulation. The concentrations of each protein type are normalised independently.

Input into the model is achieved by injecting specific concentrations of free or extra TF-proteins into the model. These free proteins have no gene associated with them and their signatures are hard-coded. Input values are encoded into the extra proteins' concentration levels. The concentration values of free proteins remain static, unaffected by regulation within the system. Their values only change when new inputs are injected, replacing the existing inputs. Output from the system is achieved by reading the concentration values of the P-proteins and interpreting them in some way. Examples of input and output to the system are given in Section 9.2.1 when describing the inverted-pendulum problem.

Regulation

Regulation remains the same in this extension of the model with one exception. Once all produced TF-protein concentrations have been updated and normalised, these concentrations are then scaled, such that, when summed with the concentrations values of the free TF-protein (inputs), they will sum to 1.0.

A formula similar to that used for TF-genes, given in Section 4.2.2 is used to calculate the expression rate of P-genes.

$$\frac{dc_{it}}{dt} = \delta(e_{it} - h_{it})$$

$\frac{dc_{it}}{dt}$ is added to the concentration of the protein, p_{it}, for both protein types. P-protein concentrations are then normalised separately from the TF-protein concentrations to sum to 1.0.

Synchronisation with the Environment

While this model accepts input signals and provides output signals, there is no direct relationship between the dynamic problem environment and the GRN. After the initial environment state has been encoded and injected into the system as free TF-proteins, a decision must be made when to stop iterating the GRN and to extract the output signals. This is required to allow the GRN to affect the problem environment and, in turn, to receive updated state information. The GRN could be allowed to iterate until a steady state has been achieved, however, as the dynamics of the GRN can be chaotic, there is no guarantee that the GRN will stabilise into a steady state, or if it does, that it will achieve this in a reasonable number of iterations. Therefore, a user defined parameter, *a sync size*, is used. A *sync*, or a synchronisation step, is a finite number of iterations of the GRN. Inputs are injected into the GRN, signalling the start of a sync. After sync iterations the output concentrations are read. Depending on the problem, a sync can represent a single time-step of the problem environment, or if the environment is a time-based simulation, a time period of some length.

4.3 Summary

This chapter provides a bearing of the current state of developmental evolution and the use of GRNs in the field of evolutionary algorithms, and specifically EC. Initially, an overview of some work using GRN models in the field of EC is given, and a number of different models are outlined. Following this a description of the Banzhaf artificial GRN model is provided, along an outline of an extension to this model by Nicolau enabling its use for EC.

With the introduction of GE, TAGs and a background in to developmental evolutionary algorithms and GRNs complete, this concludes the first section of the thesis, i.e., the background information section. The next section explores a novel extension of the linear GE representation which makes use TAGs, and how this extension might affect the GE algorithm. The properties and behaviour of this new TAG-based representation are examined on common benchmark problems and are compared to those of canonical GE.

In addition, the use of this TAG-based representation in the field of developmental evolutionary algorithms is explored. Specifically, the proceeding chapter introduces this TAG-based extension of GE, Tree-Adjoining Grammatical Evolution.

Part II

Exploration

Chapter 5

Tree-Adjoining Grammatical Evolution

In this chapter, the novel Tree-Adjoining Grammatical Evolution (TAGE) [140, 141, 142, 143] is proposed and developed in order to make use of TAGs in GE. This chapter describes the required modifications to the GE algorithm in order to make use of TAGs. Firstly, a discussion is provided on TAG composition operations and how they might be used in conjunction with the linear genotype used by GE. Following this, a genotype to phenotype mapping algorithm which makes use of TAGs is described in detail. Other components of the GE algorithm are then discussed in the context of their use in TAGE.

The chapter proceeds with a discussion on the use of tree composition operations for linear mapping in Section 5.1. An algorithm for mapping the linear GE genotype to phenotype using TAGs is provided in Section 5.2, along with a worked example of this algorithm. Section 5.4 discusses some of the limitations of the TAGE representation presented in this thesis. Finally, a summary of the chapter is provided in Section 5.5.

5.1 Tree Composition Operations for Linear Mapping

While TAGs can construct derivation trees using two different composition operations, adjunction and substitution, both of which are described in Section 3.1.1, this thesis makes exclusive use of adjunction. While TAGs which do not make use of the substitution operator are equivalent to those that do, the use of substitution operations allows for a much more compact representation, with fewer elementary trees. This is discussed further in Section 5.4.

An interesting and powerful feature of TAGs is that they have the ability to guarantee complete derivation trees from which valid phenotypes may be extracted at every stage of the derivation process. This property is known as feasibility and is introduced in Section 3.1. This property guarantees that a TAG derivation tree of *any size* will produce a syntactically valid sentence, as defined by the grammar.

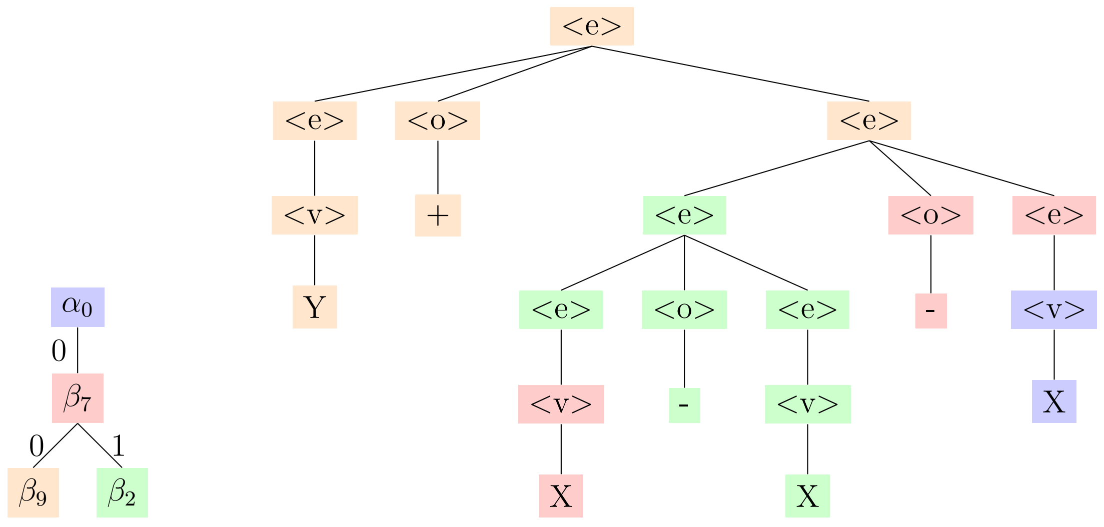
In order to guarantee feasibility when mapping from a GE chromosome of finite length using a TAG, the use of the substitution operation must be excised. If both composition operations are to be used, the chromosome would require partitioning, with the first partition mapping adjunction operations and the second mapping substitution operations. The amount of substitution operations required to complete the tree is dependent upon the quantity of adjunction operations applied. As there is a finite number of codons in the chromosome, no guarantee can be made that the second partition will be of a sufficient length to map the required amount of substitution operations to complete the tree. Alternatively, a quasi-diploid chromosome approach could be used, where a second chromosome is used for the sole purpose of mapping substitution operations. While this approach removes the need for a chromosomal partitioning scheme, it remains impossible to guarantee that there will be sufficient codons to map the requisite number of substitution operations to complete the tree, and hence, impossible to ensure feasibility.

As mentioned in Section 3.1.2, the tree-based TAG3P+ [82] addressed the problem of maintaining feasibility by regarding substitution as an *in-node operation*, attaching a list of initial trees (called *lexemes*) to each derivation node. These lexemes are preserved during operations which affect only adjunction operations such as crossover, with the lexemes themselves being operated upon separately. This approach is made possible as a result of the tree-based representation used by TAG3P+, while with a linear genotype, such as GE, the limitations outlined in the paragraph above apply.

The exclusive use of the adjunction operation requires the TAGs utilised in this thesis to be fully anchored LTAGs. That is to say that, with the exception of the foot node, all branches of the generated auxiliary trees are fully expanded with terminal symbols labelling all leaf nodes. In doing so, this allows the adjunction operation to take a complete tree as input, and return a complete tree as output. The output tree is composed of an auxiliary tree adjoined to the input tree. This approach guarantees complete derivation trees before and after each adjunction operation, without the use of the substitution operation, ensuring feasibility. For example, initial and auxiliary trees can be seen in the sample grammar presented in Figure 3.3. This design decision introduces some limitations to the system which are discussed in Section 5.4.

5.2 Genotype-Phenotype Mapping

As defined in Section 3.1, TAGs make use of tree composition operations, combining partial or elementary trees together in the construction of their structured sentences. As discussed in Section 3.1.2, TAGs differentiate between the derivation and derived tree. While the resulting TAG derived tree (Figure 5.1b) is equivalent to the CFG derivation tree, TAG derivation can be more compactly described by using a TAG derivation tree (Figure 5.1a). As is described in Section 3.1.2, the nodes of a TAG derivation tree are labelled with elementary trees, and each of the edges between derivation nodes is labelled with an address. This address provides the location of a particular node within the elementary tree labelling the parent derivation node of that edge. It is at this location that the auxiliary tree labelling the child node of the edge is to be applied using a composition operator. Applying



(a) TAG derivation tree.

(b) TAG derived tree.

Figure 5.1: An example of a TAG derivation tree and its respective derived tree. The derivation tree (a) defines the adjunction operations which produce the derived tree (b). The nodes in (a) are labelled with elementary trees with each edge declaring an adjunction operation between parent and child. Each edge is labelled with the address of a node in the parent tree where the adjunction will occur.

each of these composition operations creates the TAG derived tree.

In TAGE, the genotype is the same as that in GE. However, while the codons are applied to the same general mapping rule

$$codonValue \bmod c = \ choice.$$

c no longer represents the number of available production choices. Rather, in place of c there is either the number of possible locations throughout the entire derivation tree where adjunction might be applied,

$$codonValue \bmod ||adjoinableAddresses|| = \ choice$$

or the number of auxiliary trees, one of which, may be adjoined to a particular location.

$$codonValue \bmod ||A'|| \;=\; choice$$

Another important difference from GE is that mapping from genotype to phenotype in TAGE will only terminate when the end of the chromosome is reached. While mapping in GE may expand all non-terminal nodes before reaching the end of the chromosome, there are always adjoinable nodes available in TAGE derivation trees.

To begin mapping, a genotype to phenotype in TAGE an initial tree is required and is chosen from the set of initial trees, I. A list of addresses of adjoinable nodes in the tree is then created. A node address is that node's index in a breadth first traversal of the elementary tree it is contained in. An adjoinable node is any node which has not already been adjoined to and is labelled with a symbol which is also the label of the root node of an auxiliary tree in A. Adjunction is restricted from occurring on foot nodes. An auxiliary tree is then chosen from A. This auxiliary tree will be adjoined to the initial tree at the chosen node. Both trees now form a TAG derivation tree of size two. With an edge going from root node, labelled with the initial tree, to child node, labelled with the auxiliary tree. This edge denotes the adjunction operation to be performed. The trees labelling each node are traversed, updating the list of adjoinable node addresses. The used address is removed and the addresses of any additional adjoinable nodes in the new auxiliary tree are added. The next node address and auxiliary tree are chosen and added to the derivation tree. All choices in this example consume a codon, and the above process continues until all codons are consumed. A worked example is presented in the following section, with each step of mapping portrayed in Figure 5.2. Pseudo-code for the mapping can be seen in Algorithm 5.1.

Algorithm 5.1 TAGE Derivation - Mapping from genotype to phenotype using a TAG, G, and a chromosome, C. A derivation tree is constructed by reading codons to select adjoinable addresses and auxiliary trees to adjoin at those addresses. Each adjunction operation requires two codons. The process stops when insufficient codons remain. The adjunctions defined in the derivation tree are then applied, and a derived tree is constructed. The phenotype is extracted from the leaf nodes of the derived tree. The application of adjunction operations is described in Section 3.1.1. Example derivation and derived trees can be see in Figure 5.2.

```
Require: $G = \{N, \Sigma, I, A, S\}$ {A TAG}
Require: $C$ {A chromosome}
  $addresses \leftarrow \emptyset$
  $codon \leftarrow C$.removeFirst
  $root \leftarrow I$.get($codon$)
  $nodeAddresses \leftarrow root$.getAdjoinableAddresses
  for all $a_i$ in $nodeAddresses$ do
    $addresses$.add(($root, a_i$))
  while $C$.size> 1 do
    $codon \leftarrow C$.removeFirst
    $node, address \leftarrow addresses$.remove($codon$)
    $type \leftarrow address$.getSymbol
    $trees \leftarrow A$.getAll($type$)
    $codon \leftarrow C$.removeFirst
    $auxTree \leftarrow trees$.get($codon$)
    $node$.adjoin($auxTree$, $address$)
    $nodeAddresses \leftarrow node$.getAdjoinableAddresses
    for all $a_i$ in $nodeAddresses$ do
      $addresses$.add(($node, a_i$))
  $tree \leftarrow root$.getDerivedTree
  $phenotype \leftarrow \epsilon$
  for all $node_i$ in $tree$.getLeafNodes do
    $phenotype \leftarrow phenotype + node_i$.getSymbol
  return $phenotype$
```

Example 2 (TAGE Mapping Example) Given the TAG G, where $\Sigma = \{$`X, Y, +, -, *`$\}$, $N = \{$`<e>,<o>,<v>`$\}$, $S =$`<e>` and I and A are as shown in Figure 3.3, derivation using the chromosome from Figure 2.2 proceeds as follows.

The first codon value, `12`, is read. This is used to select an initial tree from I, by using $||I||$, which is in this case is `2`. Utilising the same mapping function as GE, `12 mod 2 = 0`, the zeroth tree from I is chosen, α_0. This tree is set as the root node of t, the derivation tree (as seen in Figure 5.2a).

Following this, a location to perform adjunction must be selected. The vector V is created of the adjoinable addresses available within all nodes (trees) contained within t. An adjoinable address in an elementary tree is the breadth first traversal index of a node. The node must be labelled with a NT symbol of which there is an auxiliary tree of that type, and where no auxiliary tree is currently adjoined at that index. In this case, $V = \{\alpha_0[0]\}$ (the zeroth node of α_0). A codon is read, `3`, and an address is selected from V using $||V||$, `3 mod 1 = 0` indicating which address to choose, $V[0]$. Adjunction will be performed at $\alpha_0[0]$, or node index `0` of tree α_0, `<e>`. Next, an auxiliary tree is chosen from A that is of the type `T`, i.e., the label of root node of the auxiliary tree is `T`, where `T` is the label of the node at which adjunction is being performed. In this case `T = <e>`. There are 12 such trees in A. Reading the next codon, `7`, and using $||A||$, `7 mod 12 = 7`, β_7 is chosen. This is added to t as a child of the tree being adjoined to, labelling the edge with the address `0` from $\alpha_0[0]$, see Figure 5.2b. The adjoinable addresses in β_7 are added to V for the next pass of the algorithm, $V = \{\beta_7[0], \beta_7[1]\}$. A codon is used to select an address, followed by another codon to select an auxiliary tree of the correct type, adding a new node and edge to the derivation tree with each pair of codons read. This process is repeated, consuming two codons with each iteration, until all remaining codons have been read. If the chromosome has an even number of codons, there will be a single codon remaining, unread, as an intron at the end of the chromosome. The resulting derivation and derived trees at each stage of this process can be seen in Figure 5.2. □

As mentioned above, unlike GE, this TAGE mapping algorithm effectively consumes the entire chromosome. This property results in a strong correlation between chromosome length and the resulting derivation tree size, and as a result, derived tree size and phenotype length also. That is to say, individuals with the same length will produce derivation trees of the same size, and depending on the variety of elementary tree sizes in the grammar, derived trees and phenotypes of similar sizes - short chromosomes result in small trees, and large chromosome result in large trees. The function of chromosome length to derivation tree size is as follows

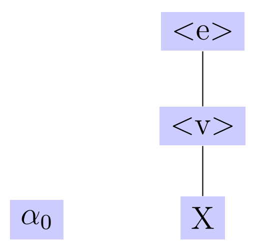
$$n = 1 + \lfloor (l - 1)/2 \rfloor$$

or written more succinctly as

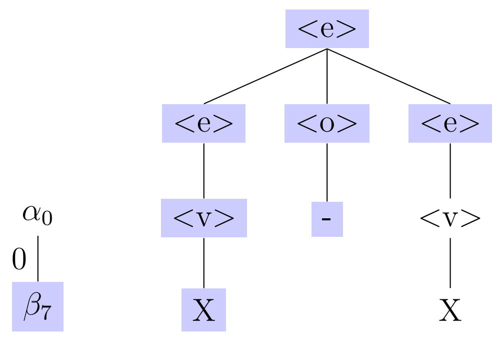
$$n = \lceil l/2 \rceil$$

where n is the number of nodes in the derivation tree and l is the length of the chromosome (a positive, non-zero, integer). One codon is used to select an initial tree, and two codons are used for each adjunction operation.

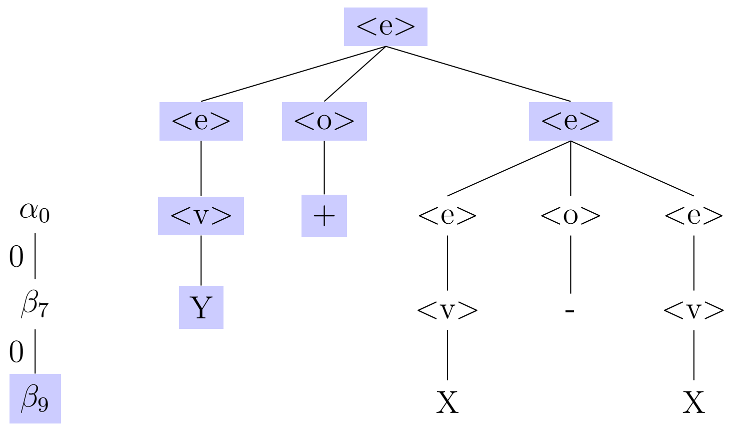
This property of derivation tree size as a function of genotype length provides easy control of derived tree size and phenotype size by simply restricting genotype length, which can be useful when addressing bloat. This relationship between genotype and phenotype lengths is explored for use for population initialisation in Section 8.2.1.



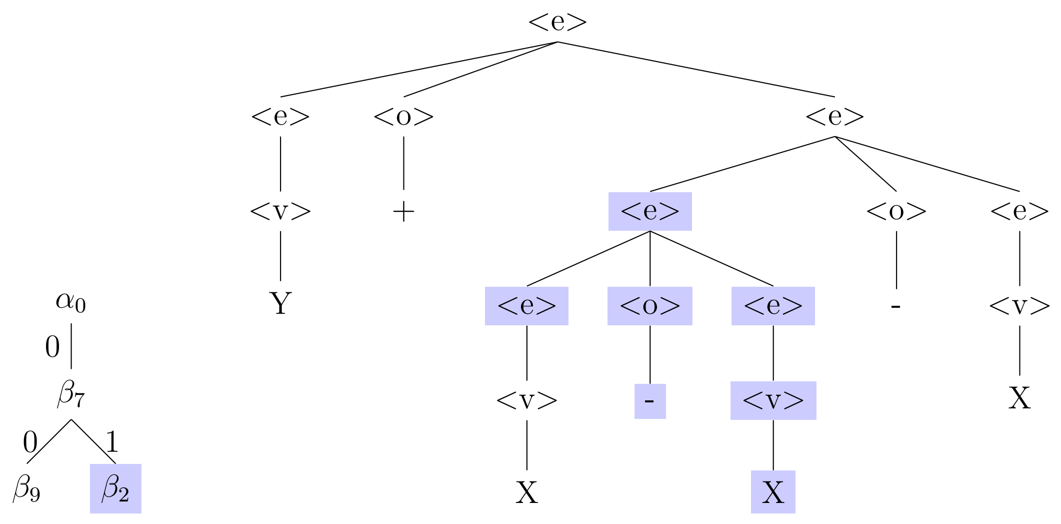
(a) Initial tree α_0.



(b) β_7 adjoined to α_0 at node address 0.



(c) β_9 adjoined to β_7 at node address 0.



(d) β_2 adjoined to β_7 at node address 1.

Figure 5.2: The derivation tree (left) and derived tree (right) throughout TAGE derivation. The shaded nodes indicate new content added at each step.

5.3 Operators

This section describes the standard GE components and how they are applied in TAGE.

Initialisation Basic initialisation used for TAGE is the same as that for GE. As both representations are based upon linear integer genomes, random initialisation, as described in Section 2.4.1, can be used. Further discussion regarding initialisation in TAGE is given in Chapter 8.

Selection and Replacement Both selection and replacement in TAGE are the same as those in GE as described in Section 2.4.2.

Variation Operations As both TAGE and GE share the same genotype, both variation operations used by canonical GE, integer-flip mutation and one-point crossover (Section 2.4.4), can be utilised by TAGE. However, due to the genotype to phenotype mapping used by TAGE, these operations can generate different effects when used.

In standard GE, the modification of a single codon in the encoding region of a GE chromosome can have multiple effects. These effects include: no change due to the use of the mod rule; changing a single terminal symbol; affecting the entire course of expansion in the derivation tree from the expansion encoded by that codon onward, through the right side of the derivation tree, due to the ripple effect. Whereas in TAGE, the same modification can bring about a different range of effects. Small effects similar to those of GE can occur: with no changes to the phenotype occurring, due to the redundancy of the mod mapping rule; the modification of a single terminal symbol by replacing an auxiliary tree with similar tree; or the modification of many terminal symbols by replacing the auxiliary with a very different tree. However, as codons in the TAGE genotype encode both for location (adjoinable addresses) and content (auxiliary trees), a modification to the chromosome which affects the number of adjunction addresses available for the next adjunction operation can induce a ripple effect (Section 2.4.4). Unlike GE, there is no directional dependence inherent in the TAGE genotype-phenotype mapping. As such, adjunctions may occur in any part of the tree at any stage of derivation. This causes the ripple effect to occur throughout the entire tree, rather than just at the non-terminal expansions which follow the affected expansion as seen in GE.

One feature of the use of these operators with the TAGE representation is that only crossover, a highly destructive operator in TAGE, can affect the size of a TAGE individual. With canonical GE, where only a portion of the genotype might be used to encode the phenotype, mutation can introduce additional non-terminals into the derivation sequence requiring the use of additional codons to terminate mapping, resulting in a larger phenotype. With TAGE, the entire genotype is used to encode the phenotype, and as such, mutation has no effect on the size of the derivation tree. Mutation can only affect the size of the phenotype if elementary trees of varying size are mutated into the derivation tree. It is only through crossover that TAGE genotypes and derivation trees can change in size.

5.4 Limitations of TAGE

This section outlines some restrictions of the TAGE representation presented by this thesis.

5.4.1 Mapping

As mentioned in Section 5.2, TAGE mapping effectively consumes the entire chromosome, with at most a single intron at the end of the chromosome - unlike GE, where mapping can utilise anything from the minimum number of codons to create the shortest phenotype, to using all codons.

This property of TAGE mapping makes fixed, same-length, genotypes unsuitable for use with TAGE, as all phenotypes produced from these genotypes will be of similar length. The effect of this can be seen in Section 8.2, and in particular in Figure 8.3.

5.4.2 Grammar Sizes

As substitution allows for a more compact representation [99], making sole use of the adjunction operation can result in the grammar becoming very large. CFGs containing non-terminal symbols which can be expanded using many different production rules can result in the TAGE representation becoming unfeasible. As Algorithm 3.1 creates all auxiliary trees from an enumeration of all possible expansions of the non-terminal symbols from root to leaf, a small increase in the number of possible expansions for any symbol can result in a large increase in the number of trees in the representation. As an example, rules from the relatively simple grammar in Figure 2.2 are modified,

```
<e>:= <e><o><e> | <v>                <e>:= <e><o><e> | <v>
<o>:= + | - | *              ->      <o>:= + | - | * | / | ^
<v>:= X | Y                          <v>:= X | Y | Z
```

adding extra production choices for the <o> and <v> symbols. When Algorithm 3.1 is applied to the modified grammar, the number of auxiliary trees in the grammar increases to from the original 12, as seen in Figure 3.3, to 30. As the grammar complexity increases the number of trees also increases. A prime example of this is seen when transforming a much more complex grammar which was used by Galvan-Lopez et al. [54]. This grammar, presented in Figure A.1, generates an auxiliary tree set of 261848 trees when applied to Algorithm 3.1.

Tree-stubs

To partially address this limitation, the sets of initial and auxiliary trees in TAGE are not generated at the beginning of the algorithm, but rather sets of elementary tree *stubs* are generated. This can greatly reduce the amount of memory needed to store the grammar. An elementary tree stub is an almost fully expanded elementary tree, with the terminal symbol leaf nodes excluded. In their place is a number representing the total number of different terminal nodes (variations) that can be attached at that point to complete the tree. For example, continuing with the sample grammar from Figure 3.3, the original 12 auxiliary trees can be reduced to two auxiliary tree stubs. The modified version of the grammar presented above which originally had 30 auxiliary trees is also reduced to just two auxiliary stubs. Taking the grammar presented in Figure A.1, the 261848 auxiliary trees are reduce to only 6452 auxiliary tree stubs which are generated and stored at the beginning of the algorithm. The process of expanding a stub into the correct complete elementary tree is described in the proceeding paragraph, as well as being outlined in Algorithm 5.2.

When choosing a tree in TAGE, the modulus operation is performed on the codon value and the number of trees available for selection. This results in a number, c, between zero and the total number of trees less one. If c has been used before, the correct tree is retrieved directly from a map of previously constructed trees. Alternatively, if c has not been seen before, then in order to find the correct stub to expand, each stub's total number of variations are summed, in order, until the sum is greater than c. Following this, the stub is completed by visiting each NT leaf node of that last stub in a depth-first manner, dividing c by the product of the variations of all the NT leaves visited so far while expanding that stub. Performing the modulus operation on this product and the number of possible variations at the current NT node results in a number between zero and the total variations possible at that node. This allows the selection of the correct terminal production to expand the current node with. The process continues until there are no additional NT nodes to expand, and the complete tree is stored in a map for later use before being returned. This process is further explained in Algorithm 5.2.

This process addresses the problem of transforming CFGs which contain many terminal production rules for the same symbol into TAGs. It does not, however, fully solve the problem of transforming highly complex CFGs into TAGs. If a CFG has a large number of non-terminal production rules for the same symbol, regardless of the number of terminal productions then the size of the generated tree sets will remain large.

5.5 Summary

This chapter provides a description of a novel method, TAGE, combining TAGs with GE in place of the more commonly used CFGs. A novel genotype-phenotype mapping algorithm is presented, allowing integer string chromosomes to be mapped to sentences using TAG. This algorithm is designed to make exclusive use of the TAG adjunction composition operation in order to ensure the feasibility property of TAGs is maintained. Some limitations of this approach are also discussed.

The next chapter provides an exploration of this new TAG-based representation for GE, TAGE, by examining its properties and behaviour on common benchmark problems and comparing them to those of canonical GE. Specifically, the next chapter investigates the effectiveness of TAGE in terms of performance.

Algorithm 5.2 Selecting a complete elementary tree from stubs

```
Require: graph {the symbol/production graph used in TAG creation}
Require: symbol {root symbol for stubs}
Require: stubs {a nonempty set of stubs rooted with symbol}
Require: choice {the codon value}

  {Get choice in the correct range}
  $choice \leftarrow choice$ mod $stubs$.sumVariations

  {Find which stub to expand}
  $sum \leftarrow 0$
  while $sum \leq choice$ do
    $sum \leftarrow sum+$ getNextStub.variations
  $stub \leftarrow$ getCurrentStub

  {While there are still nodes to expand}
  {Get the next node and count its variations}
  $product \leftarrow 1$
  while $stub$.hasNTLeafNodes do
    $node \leftarrow stub$.getNextNTLeafNode
    $vars \leftarrow node$.getVariations

    {Graph edges are labelled with productions}
    {Get the productions with only terminal symbols}
    $nrprods \leftarrow graph$.getNonRecursiveEdges(node.label)
    $tprods \leftarrow nrprods$.getTerminalProductions

    {First divide by products to discard information for already expanded nodes}
    {Then modulus by the number of variations at that node}
    $production \leftarrow tprods$.get(($choice/product$) mod $vars$)

    {Add the new production to the tree}
    {Multiply so the next node will discard the correct information}
    $node$.addChildren($production$)
    $product \leftarrow product * vars$
```

Chapter 6

A Comparison of TAGE and GE

The intention of this thesis is to analyse and understand the effect of using TAGs, in conjunction with the linear chromosome of GE, as a representation for evolutionary search. Having introduced TAGE, a novel extension of GE using TAGs, this chapter now aims to examine the hypothesis, of whether this incorporation of TAGs into the standard GE algorithm will affect the performance of the algorithm. Some of the experiments presented in this chapter are based on work published by Murphy et al. [140].

While TAGE and standard GE share a common genotype, the linear chromosome, representation in GE is composed of both a genotype and a mapping. It is this mapping, the relationship between the search space and the solution space, that has been modified in TAGE. In order to investigate the effect of this change in representation, experiments are presented in this chapter comparing the performance of GE and TAGE on a number of benchmark problems. As GE is a stochastic algorithm, each experiment is performed a number of times and the average values are examined.

The remainder of this chapter is comprised of two main sections. Section 6.1 outlines experiments which compare canonical GE and TAGE, in terms of performance, on a number of benchmark problems. The different problems are described in Section 6.1.3 and the results of both TAGE and GE are given. In the following discussion, Section 6.1.4, a number of differences between the representations are identified. Section 6.2 goes on to examine these differences in more detail, and further experiments are presented which analyse the effect that of each of these representational differences has on the ability of the algorithm to perform. Finally, a summary of the chapter is provided in Section 6.3.

6.1 A Comparison of Performance

Chapter 5, describes TAGE, an extension of the GE algorithm which incorporates the use of TAGs into the representation. TAGs are interesting as they are more powerful than CFGs which are usually used with GE. This section investigates the effect of this change in representation, from CFG to TAG, on the GE algorithm and presents experiments which examine the following general null hypothesis:

H_0 : There is no difference in performance when making use of TAGs in GE

with the accompanying alternate hypothesis:

H_1 : There is a difference in performance when making use of TAGs in GE

A comparison between TAGE and standard GE is performed on a number of benchmark problems from classic GP literature, and the best and mean population fitness values are recorded. H_0 is tested by comparing the distributions of best fitness values produced by both representations at the final generation of each run using a statistical test.

The experimental setup is given in Section 6.1.2 with a brief discussion on the choice of statistical tests in Section 6.1.1. Each benchmark problem is described and the results of the experiments are presented in Section 6.1.3. Finally, analysis and discussion of the experimental results are provided in Section 6.1.4.

6.1.1 Statistical Tests

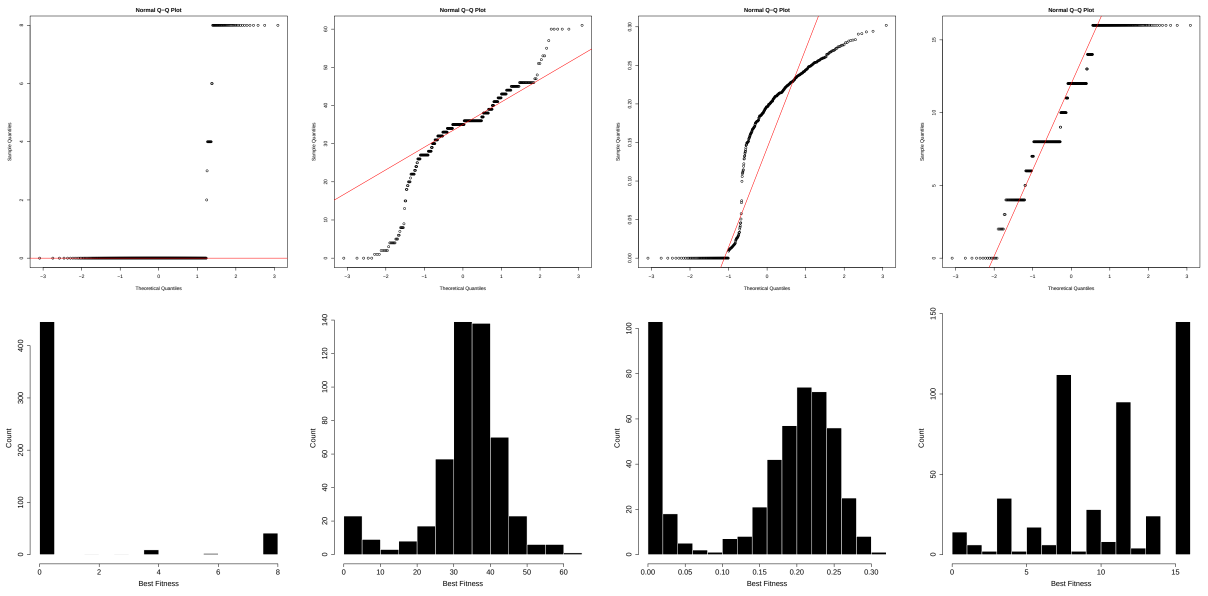
There are two main classes of statistical tests for comparing populations/data sets, parametric and non-parametric tests. Parametric methods deal with the estimation of population parameters, i.e., parameters of the data such as the mean, variance, standard deviation, and distribution. As such, parametric tests make assumptions about these properties of the data, e.g., many parametric tests assume that the data has a normal distribution. Non-parametric methods, on the other hand, make no such assumptions regarding these parameters, allowing the examination of the difference between groups and/or variables whose parameters are unknown. Non-parametric tests are considered less powerful than parametric tests, however by using large sample sizes they can be considered as powerful.

A sample of the data produced by the experiments performed in this chapter can be seen in Figure 6.1. This data is plotted on Q-Q plots and the result of the Shapiro-Wilk test for normality is shown. For each of the problems, it can be seen that the data points do not follow the expected line for a normal distribution and that the Shapiro-Wilk test p-value is well below the threshold value of 0.05, rejecting the null hypothesis that the population is normally distributed.

As a result of this, for experiments throughout this work, distributions of various properties of the individuals produced by both representations at the final generation of each run are compared using the Mann-Whitney U, two-tailed, non-parametric test. This tests whether the population distributions are identical without assuming them to follow the normal distribution. A 5% level of significance is used for this test throughout.

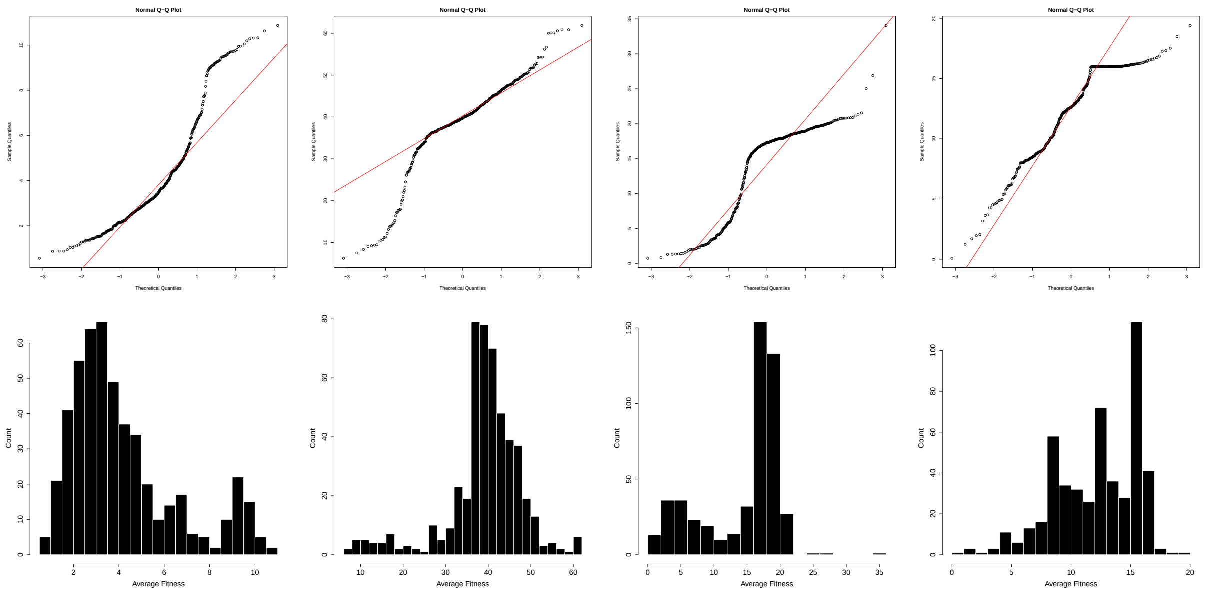
6.1.2 Experimental Setup

Five hundred independent runs are performed with GE and TAGE using the GEVA software [180] on each of the problems outlined below. The evolutionary parameters adopted



$p < 2.2\text{e-}16$ $p < 2.2\text{e-}16$ $p < 2.2\text{e-}16$ $p < 2.2\text{e-}16$

(a) Best Fitness: Even Parity, Santa Fe, Symbolic Regression and Six Multiplexer



$p < 2.2\text{e-}16$ $p < 2.2\text{e-}16$ $p < 2.2\text{e-}16$ $p = 7.77\text{e-}15$

(b) Mean Fitness: Even Parity, Santa Fe, Symbolic Regression and Six Multiplexer

Figure 6.1: Q-Q plots and histograms of GE results for both the best and average population fitness of each problem at the end of each run. The result of the Shapiro-Wilk test for normality (p-values) for each problem is also included. From this figure it is clear that the populations examined have non-normal distributions, and as such, non-parametric statistical tests are used.

Table 6.1: GE parameters.

Parameter	Value
System	GEVA v1.1 (extended)
Random Number Generator	Mersenne Twister MT199973
Generations	200
Population Size	100
Initialisation	Random
Initial Chromosome Size	15
Max Chromosome Wraps	0
Replacement Strategy	Generational
Elitism	10 Individuals
Selection Operation	Tournament
Tournament Size	3
One Point Crossover Probability	0.9
Integer Mutation Probability	0.02

for all of the experiments are presented in Table 6.1.

A short initial chromosome length of 15 is used. Unlike standard GE mapping whereby the size of the encoding region of the chromosome varies, i.e., the number of codons used to produce the phenotype, TAGE mapping makes use of the entire chromosome. Initialising with a short chromosome ensures that TAGE has the opportunity to explore short solutions to a problem if they exist. The value used, 15, was chosen arbitrarily.

Each run is evolved for 200 generations, enabling longer solutions to be explored if needed through chromosome growth by means of single-point crossover. Wrapping, as described in Section 2.3.1, is disabled for all experiments, and indeed for all experiments presented throughout this thesis. The grammars used for each problem are shown in Figure 6.2. These CFGs are transformed into TAGs by Algorithm 3.1 for use with TAGE.

```
<prog>   ::= <expr>
<expr>   ::= <expr> <expr> <op>
           | ( <expr> <expr>  <op> )
           | <var>
           | ( <var> ) <pre-op>
<pre-op> ::= !
<op>     ::= "|" | & | ^
<var>    ::= d0 | d1 | d2 | d3 | d4
```

(a) Even Five Parity

```
<prog>      ::= <code>
<code>      ::= <line> | <code> <line>
<line>      ::= <condition> | <op>
<condition> ::= if(food_ahead()==1) { <opcode> }
                else { <opcode> }
<op>        ::= left(); | right(); | move();
<opcode>    ::= <op> | <opcode> <op>
```

(b) Santa Fe Ant Trail

```
<prog> ::= <expr>
<expr> ::= ( <op> <expr> <expr> ) | <var>
<op>   ::= + | - | *
<var>  ::= x0 | 1.0
```

(c) Symbolic Regression

```
<prog> ::= <B>
<B>    ::= <B> <B> & | <B> <B> "|"
         | <B> !
         | <B> : <B> ? <B> #
         | a0 | a1 | d0| d1 | d2 | d3
```

(d) Six Multiplexer

Figure 6.2: The grammars, in Backus-Naur form, for each of the benchmark problems. Each grammar is transformed into a TAG by Algorithm 3.1 for use with TAGE. These TAGs are presented in Figure B.1- B.4.

6.1.3 Problems and Results

Even Five Parity

The aim of this problem [113, p. 529] is to successfully recreate the five input even-parity Boolean function. This function should return a set bit (true) if an odd number of input bits are set, making the total number of set bits even. Otherwise, the function should return an unset bit (false). Generated functions are evaluated on all possible 2^5 test cases. The fitness, or error, which is minimised, is calculated as the number of incorrect test cases. The grammar used for this problem is presented in Figure 6.2a. Mean best fitness and average fitness plots can be seen in Figure 6.3. The mean best fitness values for GE and TAGE are 0.762 and 0.032 respectively. The Mann-Whitney U p-value is 1.03e-10, below the threshold value of 0.05, thus rejecting the null hypothesis H_0 for this setup. Interestingly, the mean average fitness values for TAGE populations on this problem are much higher than those of GE. This can be seen both in Table 6.2 and the mean average population fitness plot in Figure 6.3. The performance values are reported in Table 6.2,

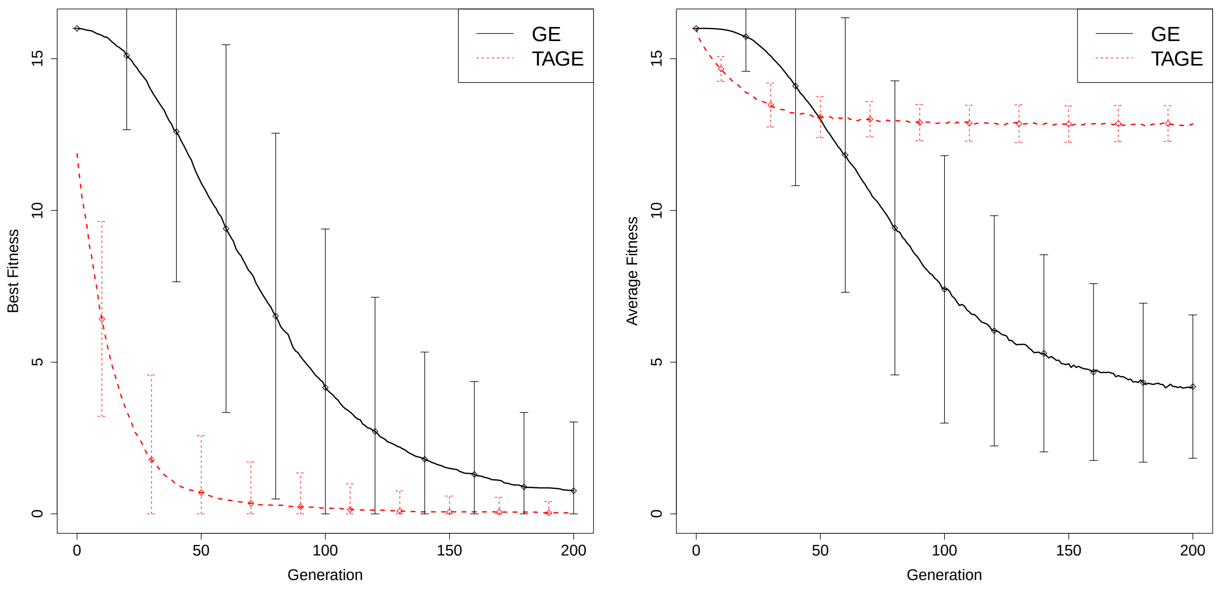


Figure 6.3: Mean best (left) and mean average population (right) fitness plots for the even five parity problem with error bars of one standard deviation.

and all p-values are presented in Table B.1.

Ant Trail

This problem [120, 113, p. 54] requires the generation of a control algorithm to guide an artificial ant. The algorithm controls the movements of the artificial ant on a 32x32 toroidal grid. The algorithm can make use of conditional statements, as well as several operations to collect 89 pieces of food located along a broken trail (the Santa Fe ant trail). These operations are: foodAhead(), enabling the ant to check if there is food in the tile directly facing it, as well as right(), left() and move(). The grammar used for this problem is presented in Figure 6.2b. The latter three operations consume one unit of energy each. The aim of the problem is that the ant will collect all food pieces along the trail before consuming all 600 units of energy. Fitness, which is being minimised, is the quantity of remaining food pieces when available energy has reached zero. Mean best fitness and average fitness plots can be seen in Figure 6.4. The mean best fitness values for

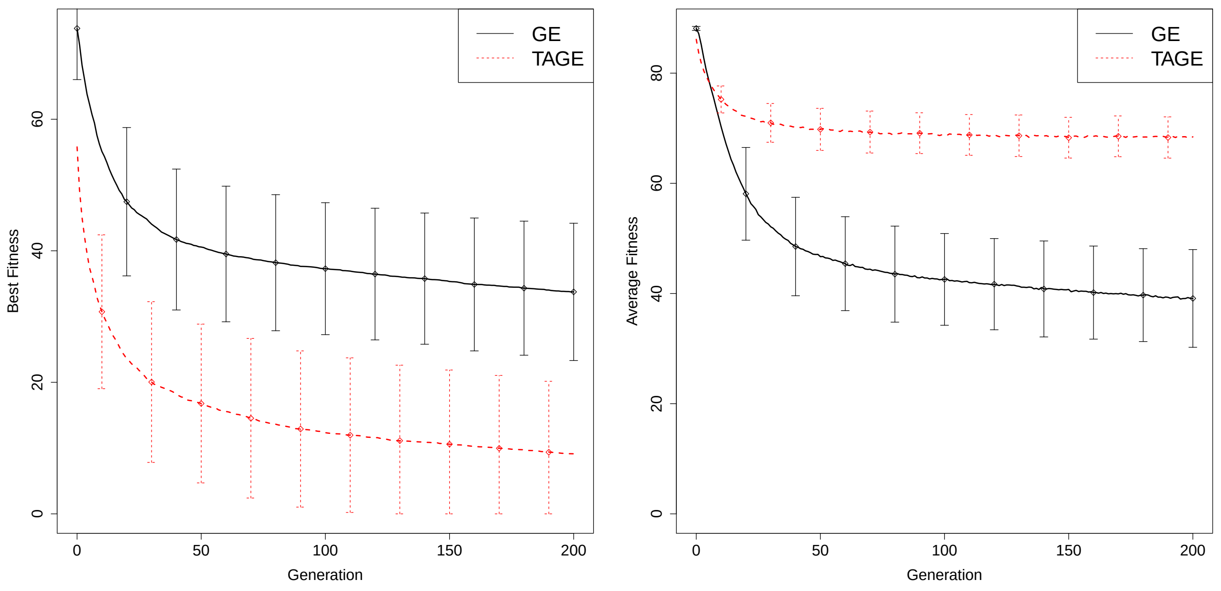


Figure 6.4: Mean best (left) and mean average population (right) fitness plots for the Santa Fe ant trail problem with error bars of one standard deviation.

GE and TAGE are 33.75 and 9.13 respectively. The Mann-Whitney U p-value of 3.73e-133 is below the threshold value of 0.05. This rejects the null hypothesis H_0 for this setup. A similar trend to the previous problem is seen in the mean average fitness values for TAGE populations on this problem. From the mean average population fitness plot in Figure 6.4 it can be seen that TAGE average population fitness is much farther from the optimum than GE. The performance values are listed in Table 6.2, and all p-values are presented in Table B.1.

Symbolic Regression

The objective of this problem is to evolve the classic quartic function, $x + x^2 + x^3 + x^4$ [113, p. 203]. Fitness is measured as the residual sum of squares (RSS) across 20 test cases drawn at initialisation from a uniform distribution in the range $[-1, 1]$. Due to imprecision in computational numerical representations and many perfect solutions of different forms existing in the search space, a successful solution is reported when the RSS is less than

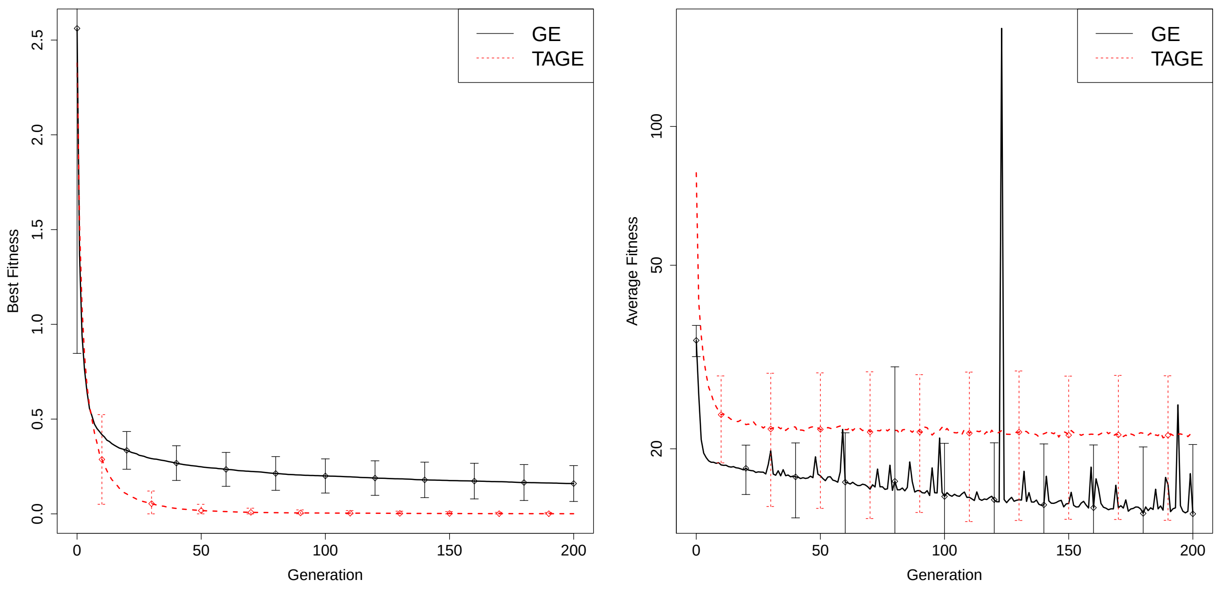


Figure 6.5: Mean best (left) and mean average population (right) fitness plots for the quartic symbolic regression problem with error bars of one standard deviation. A log scale is used on the y axis of the mean average population fitness plot to help account for the outlier. The fitness spikes are caused by exceptionally large GE phenotypes being evaluated. There are no visible fitness spikes for TAGE as TAGE chromosomes don't grow as quickly as GE chromosomes.

10^{-10}. The grammar used for this problem is presented in Figure 6.2c. Mean best fitness and average fitness plots can be seen in Figure 6.5. The mean best fitness values for GE and TAGE are 0.1599542 and 0.0007011973 respectively. The Mann-Whitney U p-value is 1.19e-96, below the threshold value of 0.05. This rejects the null hypothesis H_0 for this setup. While the TAGE best fitness values seen in Figure 6.5 are lower than those of GE, the mean average population fitness is again farther from the optimum. The performance values are listed in Table 6.2, and all p-values are presented in Table B.1.

Six Multiplexer

This problem requires the generation of a correct two address and four data bit Boolean multiplexer function [113, p. 169]. For each of the 2^6 test cases, the enumeration of all

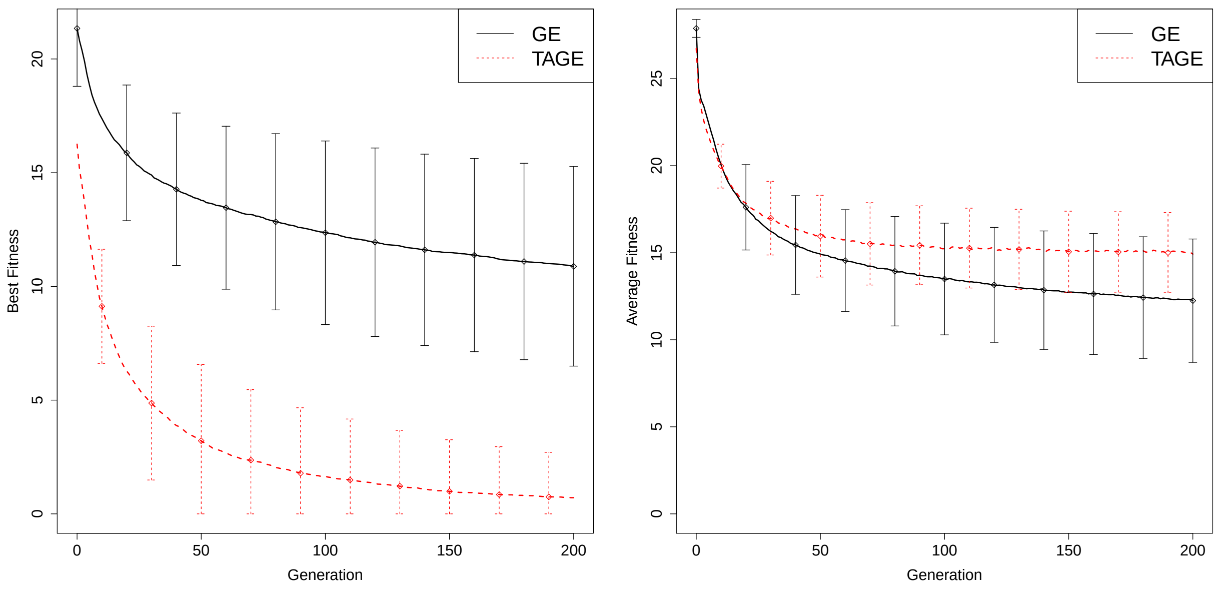


Figure 6.6: Mean best (left) and mean average population (right) fitness plots for the six multiplexer problem with error bars of one standard deviation.

possible values for the six bits, the generated function should return the value of one specific data bit. The data bit whose value aught to be returned is determined by the two address bits. Acting as a two digit binary number, these bits allow one of the four outputs to be selected. Fitness is calculated as the number of incorrect test cases, with a perfect fitness of zero for a solution returning the correct value for all 64 test cases. The grammar used for this problem is presented in Figure 6.2d. Mean best fitness and average fitness plots can be seen in Figure 6.6. The mean best fitness values for GE and TAGE are 10.884 and 0.71 respectively. The Mann-Whitney U p-value is 8.78e-160, below the threshold value of 0.05, rejecting the null hypothesis H_0 for this setup. This problem also displays the trend of TAGE populations being, on average, farther from the optimum. The performance values are listed in Table 6.2, and all p-values are presented in Table B.1.

Table 6.2: A comparison of results obtained by GE and TAGE across the benchmark problems. The mean best and mean average fitness values across 500 runs at the final generation, correct to four decimal places, and the number of successful runs are reported. Mann-Whitney U tests are performed on best fitness data with shaded cells representing a p-value < 0.05, rejecting H_0 for that setup. All p-values are given in Table B.1.

	Mean Best Fitness (Standard Dev.)	Mean Avg. Fitness (Standard Dev.)	Successes (500)
Even five parity			
GE	0.762 (2.2662)	4.1941 (2.3622)	446
TAGE	0.032 (0.3567)	12.852 (0.6104)	496
Santa Fe ant trail			
GE	33.75 (10.4377)	39.1067 (8.873)	5
TAGE	9.13 (10.6576)	68.4369 (3.6640)	224
Symbolic regression			
GE	0.16 (0.0946)	14.4539 (5.9742)	79
TAGE	0.0007 (0.0057)	21.4611 (7.3683)	489
6 Multiplexer			
GE	10.884 (4.3874)	12.2464 (3.5420)	14
TAGE	0.71 (1.9103)	14.9194 (2.3139)	423

6.1.4 Discussion

Taking the mean best fitness values and the results of the Mann-Whitney U tests performed, which are reported in Table 6.2, into account, the null hypothesis H_0, that there is no difference in performance when making use of TAGs in GE, can be rejected for this comparison.

Compared to standard GE, TAGE performs better on the experiments described in this section. Table 6.2 shows that of the four problems tested, TAGE finds more solutions than GE which achieve perfect fitness values across the 500 runs. This performance boost is reiterated when examining the mean best fitness at the end of the 200 generations. In each case, TAGE outperforms GE. Additionally, this is reflected by the mean best fitness plots in Figure 6.3 - Figure 6.6, whereby TAGE repeatedly shows an ability to find fitter solutions in fewer generations than GE, effectively searching the solution space more efficiently. Additionally, while no direct examination of bloat is performed on TAGE, from Figure 6.7 for the symbolic regression problem (and Figure B.7 for all problems), it can be seen that even though TAGE trees are larger, the size of these trees appears to level off as the generations progress, a trend which is not usually seen in bloat prone representations. Bloat results in tree size increasing throughout the generations.

A number of differences between the GE and TAGE can affect their ability to search when compared using similar settings. The most apparent of these differences from the comparison performed is as a result of the size of the basic structures used by each representation to construct their respective derivation trees. In standard GE, using a CFG, individual terminal and non-terminal symbols are combined when constructing a derivation tree. However, in TAGE, elementary trees, which are themselves composed of groups of terminal and non-terminal symbols, are combined to create a derivation tree. The result of this is that TAGE has a much more compact genotype in terms of length, with each TAGE codon having coding for a larger part of a phenotype than a GE codon. Also, the same number of TAGE codons mapping to much larger derivation trees, and hence, phenotypes than GE. This removes any bias present in the grammar with regards to generated tree size, with the chromosome length being solely responsible for tree and phenotype size. This puts the onus onto the method of initialisation and the variation operations to ensure chromosome length can be sufficiently varied during evolution.

This difference is apparent when the size of the derivation trees generated over the 500 runs for the quartic symbolic regression problem are observed in Figure 6.7. Both the initial average TAGE derived tree size (number of nodes) and average tree depth are greater than those of GE for the same initial chromosome size, and they remain larger for the 200 generations. Indicating that for a fixed initial chromosome length, the generated TAGE trees and phenotypes are larger than those of GE. This trend is repeated across three of the four problems (Figure B.7 and Figure B.8). It is possible that more solutions to these

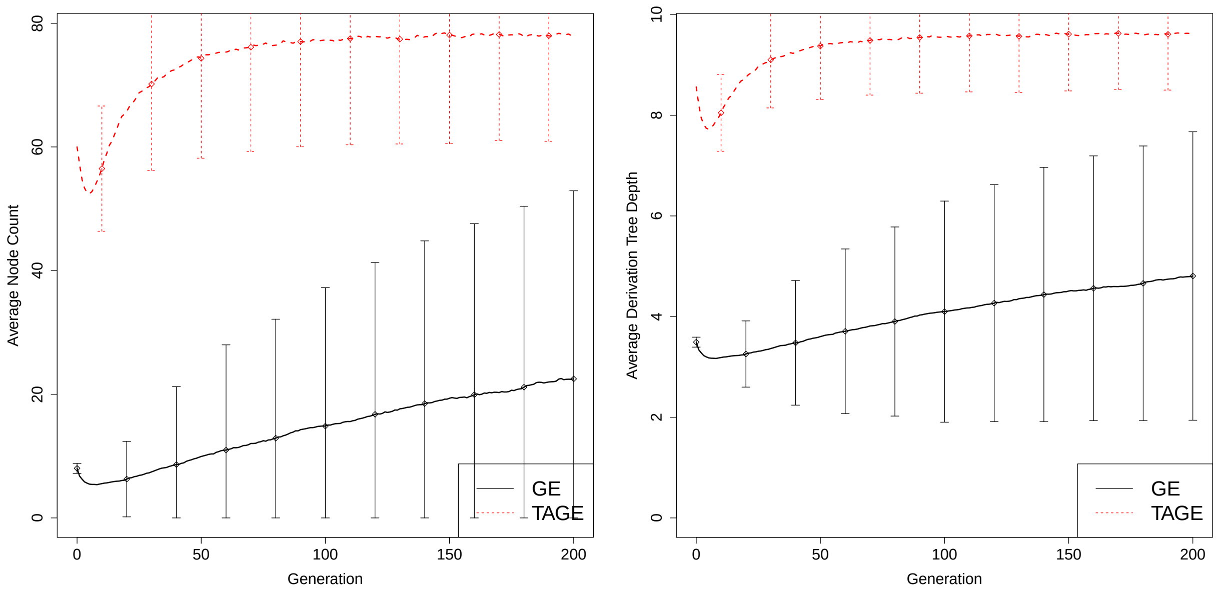


Figure 6.7: Mean node count (left) and mean tree depth (right) plots for GE derivation trees and TAGE derived trees for the quartic symbolic regression problem with error bars of one standard deviation. The node count and depth plots for all problems are provided in Figure B.7 and Figure B.8 respectively.

problems lie within the region of larger tree sizes, and that this difference in representation may be advantageous to TAGE for the experiments presented here. This is investigated further in Section 6.2.1.

The problem for which this trend did not hold is the even parity problem. This result may be explained by examining the grammar in Figure 6.2a. This grammar is an explosive grammar [63], which, in conjunction with the large chromosome size shown in Figure B.5a, would allow the large tree growth displayed by GE when solving this problem.

Another representational difference which can affect the ability to search is the size of the effective, or encoding region of the chromosome. In canonical GE, the genotype to phenotype mapping algorithm consumes codons until there are no remaining non-terminal leaf nodes in the derivation tree. The number of codons that are read from a random chromosome and the size of the resulting derivation trees and phenotypes are, therefore, highly dependent on the grammar [63]. As such, the ratio of encoding to non-coding regions can affect the effectiveness of GE's variation operations. While operators operating in the non-coding region of a GE chromosome can allow movement along fitness neutral pathways, overuse of operators in this region can hamper search. Single-point crossover, in particular, can be detrimental when allowed to operate in the non-coding region. Crossover can cause chromosomal growth, and if this growth occurs in the non-coding region without affecting fitness, then the likelihood of crossover recurring in this region increases, as crossover points are chosen uniformly. This can cause a runaway effect in terms of total chromosome length and the ratio of non-coding to encoding region sizes. In TAGE, this problem never arises as the genotype to phenotype mapping makes use of the entire chromosome, and there is no non-coding region. This can be seen for the quartic symbolic regression problem in Figure 6.8, and for all other problems in Figure B.5 and Figure B.6. A side-effect of this feature is that, unlike in GE, the concept of an invalid or non-terminating individual doesn't exist when using the TAGE representation due to the feasibility property of TAGs. As a valid TAGE derivation, and therefore derived, tree is produced from the first codon, and is guaranteed to be valid after each successive adjunction operation, it is not possible to produce an incomplete TAGE derived tree.

Interestingly, a surprising observation about TAGE is apparent from these results. While TAGE outperforms GE in terms of mean best fitness at the end of the runs and the number of perfect solutions found by these experiments, the mean average fitness values produced by TAGE are consistently worse than those of GE. That is to say, even though TAGE outperforms GE, the whole TAGE population, on average, appears to maintain a worse fitness than that of GE. While GE populations tend to converge towards the most fit individuals, TAGE populations do not. This property of TAGE is apparent in Figure 6.3 - Figure 6.6 and in Table 6.2. This property could equate to better diversity within the population which has been shown in the past to lead to improved performance by avoiding premature convergence to local optima. Further investigation into this is given

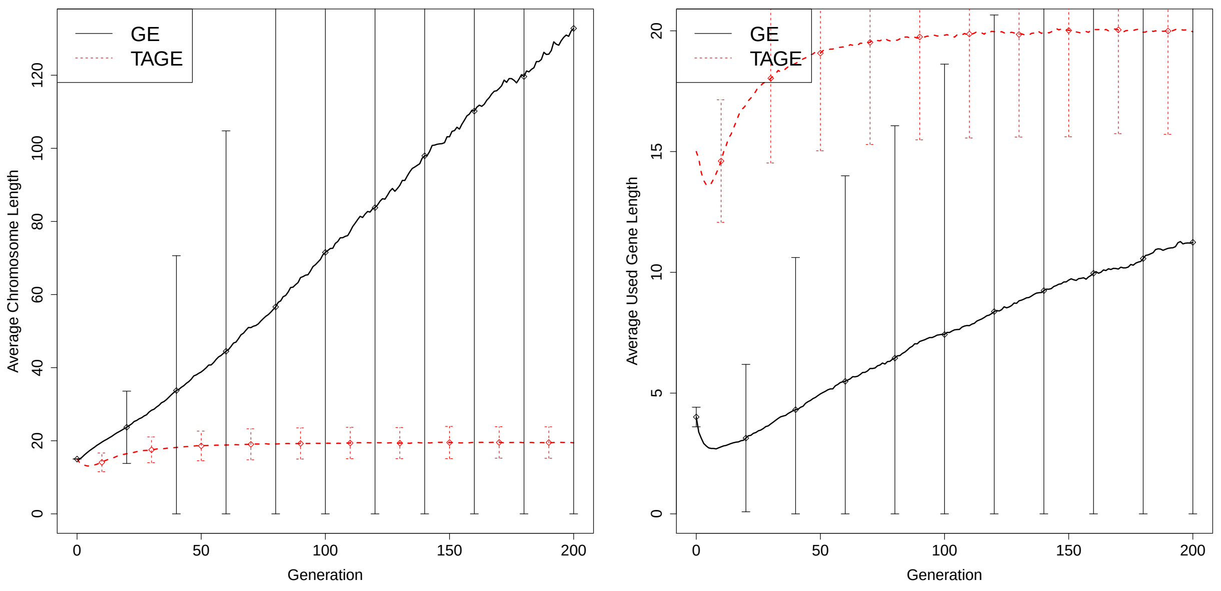


Figure 6.8: Mean chromosome length (left) and mean effective chromosome length (right) plots for GE derivation and TAGE derived trees for the quartic symbolic regression problem with error bars of one standard deviation. The mean chromosome length and mean effective length plots for all problems are provided in Figure B.5 and Figure B.6 respectively.

in Chapter 7.

6.2 Analysis of Representational Differences

Section 6.1.4 highlights differences between the GE and TAGE representations. Such differences, inherent to each representation, can make a fair comparison between the two difficult. However, by taking account of and examining the effect of each difference, further insight may be provided into the total effect produced by the change in representation from CFGs to TAGs.

This section outlines some of these differences and examines their effect on the comparison of performance, i.e., how the differences may be of benefit to TAGE over GE or vice versa. Firstly, the importance of the initial length of chromosome is examined in the context of GE and TAGE, and how it might affect search when used with random initialisation. Next, the topic of invalid individuals is examined, a property of GE not present in TAGE. Additionally, the potency of the crossover operator is examined, and how differences in the representation such as non-coding regions in the chromosome which are seen in GE and not TAGE can affect the behaviour of common operators such as single-point crossover.

6.2.1 Initial Chromosome Length

While the experiments performed in Section 6.1 make use of a common initial chromosome length of 15 for both GE and TAGE, differences in the two representations, e.g., size of the basic derivation structure and properties of the mapping procedure, can yield very different behaviours from this common starting condition. An example of this can be seen in Figure 6.7 where initial derivation tree sizes are far larger for TAGE than GE. To investigate the effect of the initial chromosome length, the comparison described in Section 6.1 is repeated here on varied initial chromosome lengths for both GE and TAGE. The original length of 15 is shown, with lengths of 30, 60 and 120 codons also used.

Experiments to examine the following null hypothesis are presented:

H_{a0} : There is no difference in performance between TAGE and GE when initialising with different length randomly generated genotypes

with the accompanying alternate hypothesis:

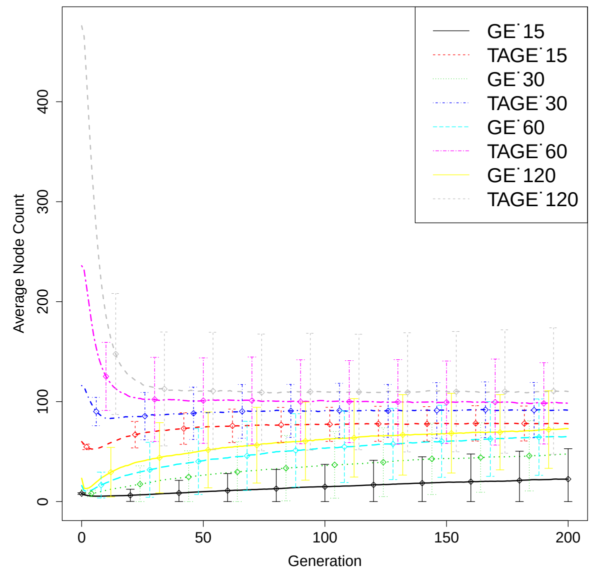
H_{a1} : There is a difference in performance between TAGE and GE when initialising with different length randomly generated genotypes

The comparison between TAGE and standard GE from Section 6.1 is performed again using a number of different chromosome lengths. H_{a0} is tested by comparing the distributions of best fitness values produced by both representations with all variants of chromosome length at the final generation of each run using the Mann-Whitney U, two-tailed, non-parametric test with a 5% level of significance.

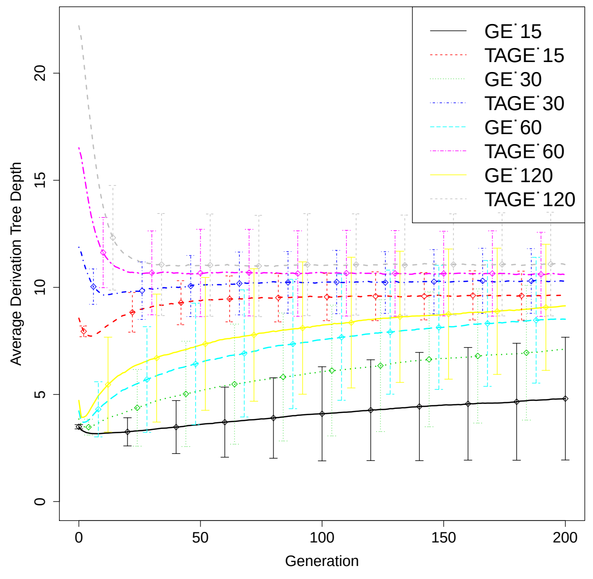
Results

From Figure 6.9a and 6.9b it is clear that for randomly initialised chromosomes, in combination with the grammars used, the lengthening of the initial chromosome has little effect on the size and the depth of the resulting GE derivation trees. However, the increase in initial chromosome length does proportionally affect these properties over the course of the runs. The increases in length provide more codons for mapping after variation operations are applied, resulting in less invalid individuals, as can be seen in Figure 6.9c, and the final row of Table 6.4. This decrease in invalids being generated can be beneficial to GE.

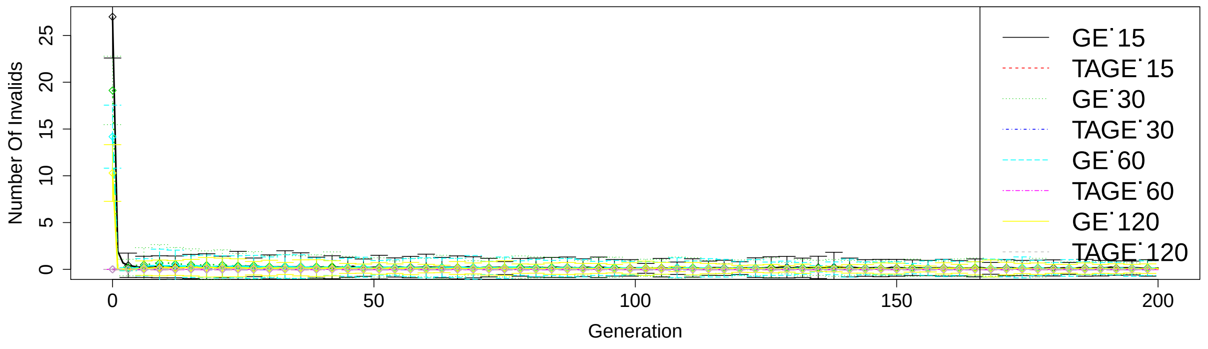
TAGE, on the other hand, which consumes the entire chromosome during mapping, with the exception of a single codon (see Section 5.2), has a proportional increase in both initial TAGE derived tree size and depth. The lengthening of the chromosome and magnitude of the increase in tree size correlates with a decrease in the mean best fitness values and the number of successful solutions found by TAGE. These values are presented in Table 6.3, with the results of the two-tailed Mann-Whitney U significance tests reported in Table 6.4. There is a statistically significant difference in the distribution of the final generation best fitness values of TAGE and GE for all but one comparison. For the even five parity problem, when initialising with chromosome lengths of 120 codons, the null hypothesis H_{a0} cannot be rejected. However, TAGE best fitness value distributions consistently have lower mean values than those of GE across all comparisons.



(a) Mean tree size



(b) Mean tree depth



(c) Mean invalid count

Figure 6.9: Plots for GE derivation trees and TAGE derived trees on the quartic symbolic regression problem with error bars of one standard deviation. The plots for all problems are given in Figure B.9, B.10 and B.11 respectively.

Table 6.3: A comparison of results obtained by GE and TAGE across the benchmark problems for varied initial chromosome lengths. The mean best fitness values across 500 runs for the final generation and the total number of successful runs are reported, correct to four decimal places.

		GE		TAGE	
		Mean Best Fitness (SD)	Successes	Mean Best Fitness (SD)	Successes
EFP	15	0.762 (2.2662)	446	0.032 (0.3567)	496
	30	0.536 (1.9121)	461	0.058 (0.5051)	491
	60	0.586 (1.9479)	454	0.076 (0.4719)	484
	120	0.352 (1.4942)	470	0.144 (0.7101)	475
SF	15	33.75 (10.4377)	5	9.13 (10.6576)	224
	30	29.038 (13.1967)	18	12.246 (12.0733)	189
	60	27.298 (13.4509)	25	14.896 (12.6748)	137
	120	27.186 (14.2802)	29	17.776 (13.7793)	113
SR	15	0.1600 (0.0946)	79	0.0007 (0.0057)	489
	30	0.0887 (0.1015)	182	0.0016 (0.0091)	480
	60	0.0541 (0.0835)	208	0.0017 (0.0083)	476
	120	0.0440 (0.0758)	218	0.0034 (0.0154)	467
6 Mux	15	10.884 (4.3874)	14	0.71 (1.9103)	423
	30	10.086 (4.4809)	27	0.742 (1.7815)	411
	60	9.738 (4.4395)	25	0.802 (1.6493)	381
	120	9.28 (4.5917)	33	1.312 (2.2461)	337

Table 6.4: Mann-Whitney U test results for best fitness values at generation 200 on varying initial chromosome size. Shaded cells represent p-values < 0.05. p-values are provided in Table B.2. Separately, each cell is labelled with T or G, indicating whether TAGE or GE has a better mean best fitness. The final row shows the mean number of invalids generated during initialisation by GE.

		GE															
		EFP				SF				SR				6 Mux			
		15	30	60	120	15	30	60	120	15	30	60	120	15	30	60	120
TAGE	15	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
	30	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
	60	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
	120	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
Invalids		27.13	19.55	14.52	10.25	30.19	7.7	0.46	0	26.99	19.12	14.18	10.3	7.01	3.09	0.97	0.15

6.2.2 The Effect of Invalids

As seen in Section 6.2.1, by increasing the initial chromosome length for random initialisation, GE can minimise the number of invalids generated during initialisation, as well as reduce the risk of invalids being produced over the course of a run. Invalids, however, remain an active difference between GE and TAGE. Variation operations in GE can introduce invalids into the population. While automatically given a poor fitness value, invalids reduce selection pressure by contending in tournament selection. This reduces the ability of GE to search efficiently. The lack of invalid individuals in TAGE could give it an advantage over GE. In order to address this representational difference, invalids are removed from GE by implementing population validation. Population validation monitors the population for invalid individuals which are produced by variation operations. If an invalid is found, that individual is removed from the population, and replaced by a copy of its valid parent individual. If crossover is used to produce the invalid individual, a single parent individual is chosen with a uniform probability. Population validation has no effect on the performance of TAGE as the representation cannot generate invalids.

Experiments to examine the following null hypothesis are presented:

H_{b0} : There is no difference in performance between TAGE and GE when replacing invalid individuals with a valid parent individual

with the accompanying alternate hypothesis:

H_{b1} : There is a difference in performance between TAGE and GE when replacing invalid individuals with a valid parent individual

The comparison between TAGE and standard GE from Section 6.2.1 is performed again with population validation enabled for GE. The best and mean population fitness values are recorded. H_{b0} is tested by comparing the distributions of best fitness values produced by both representations at the final generation of each run using the Mann-Whitney U, two-tailed, non-parametric test with a 5% level of significance.

Results

The effect of population validation on the number of invalid individuals in the population for the quartic symbolic regression problem can be seen in Figure 6.10, showing that invalid individuals after the first generation are completely removed. The benefit of replacing invalids with parent individuals is inconclusive for most of the problems examined, see Table 6.5, with GE improving marginally on some problems and performing worse on others. However, a notable improvement is seen by GE on the symbolic regression problem. The results of the two-tailed Mann-Whitney U significance tests are reported in Table 6.6. There is a statistically significant difference in the distribution of best fitness values of TAGE and GE for all but one problem, the even parity problem. For this problem, there is no significant difference on 25% of the comparisons. In particular, those of GE with initial lengths of 30, 60 and 120 codons compared to TAGE with an initial length of 120, as well as GE with length 30, compared to TAGE with length 60. For all combinations of initial lengths, with population validation enabled, TAGE produces distributions of end of run best fitness values with better mean values than those of GE.

6.2.3 Operator Effectiveness

While both GE and TAGE share the same genotype, enabling the use of the same variation operations, the effect of those operations over the course of a run vary due to other properties of the representations. For example, the encoding region of a TAGE chromosome uses up the entire chromosome, with at most one intron at the end of the chromosome,

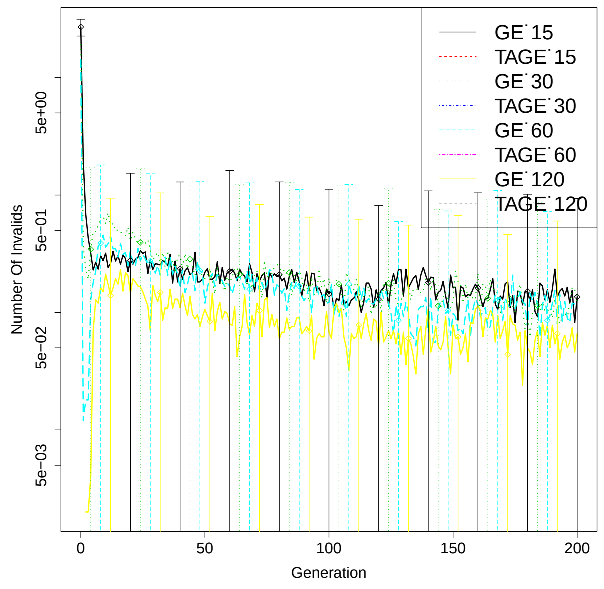


Figure 6.10: Mean invalid individual count without validation plot for GE and TAGE for the quartic symbolic regression problem with error bars of one standard deviation. With population validation, invalids are eliminated from all but the initial generation. The mean best fitness and mean invalid individual count plots for all problems are provided in Figure B.12 and Figure B.13 respectively.

Table 6.5: A comparison of results obtained by GE and TAGE across the benchmark problems with population validation enabled. The mean best fitness values across 500 runs for the final generation and the total number of successful runs are reported, correct to four decimal places.

		GE		TAGE	
		Mean Best Fitness (SD)	Successes	Mean Best Fitness (SD)	Successes
EFP	15	0.652 (2.0726)	451	0.032 (0.3567)	496
	30	0.36 (1.5860)	473	0.058 (0.5051)	491
	60	0.394 (1.5361)	464	0.076 (0.4719)	484
	120	0.308 (1.2650)	465	0.144 (0.7101)	475
SF	15	32.254 (11.9859)	10	9.13 (10.6576)	224
	30	28.222 (13.3298)	27	12.246 (12.0733)	189
	60	26.336 (13.6756)	31	14.896 (12.6748)	137
	120	27.758 (13.7739)	26	17.776 (13.7793)	113
SR	15	0.0821 (0.1033)	222	0.0007 (0.0056)	489
	30	0.0480 (0.0849)	260	0.0015 (0.0091)	480
	60	0.0367 (0.0739)	269	0.0017 (0.0083)	476
	120	0.0308 (0.0625)	271	0.0034 (0.0154)	467
6 Mux	15	10.544 (4.2563)	8	0.71 (1.9103)	423
	30	9.794 (4.3747)	23	0.742 (1.7815)	411
	60	9.614 (4.6592)	29	0.802 (1.6493)	381
	120	9.164 (4.5982)	27	1.312 (2.2461)	337

Table 6.6: Mann-Whitney U test results for best fitness values at generation 200 with varied initial chromosome size with population validation enabled. Shaded cells represent p-values < 0.05. p-values are given in Table B.3. Each cell is labelled with T or G, indicating whether TAGE or GE has a better mean best fitness. The final row shows the mean number of invalids generated during initialisation by GE.

		GE															
		EFP				SF				SR				6 Mux			
		15	30	60	120	15	30	60	120	15	30	60	120	15	30	60	120
TAGE	15	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
	30	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
	60	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
	120	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T

whereas the encoding region of a GE chromosome can be of any size. This property has an effect on the variation operations. While both operations, mutation and crossover, can occur in the non-coding region without affecting either the phenotype or fitness, crossover alone will affect the size of this region when occurring within it. If the size of the non-coding region increases, the probability of an operator being reapplied within this region also increases. This process, reoccurring with increasing probability each time the region grows, can cause the region to reach extreme sizes, becoming much larger than the encoding region. As the non-coding region grows, the effectiveness of mutation and crossover as search operators is reduced. TAGE chromosomes do not display this type of growth. As crossover is the only operator capable of modifying the length of a TAGE chromosome and there is no non-coding region in TAGE, changes in length of chromosome due to crossover have a greater chance of affecting fitness, with selection pressure operating on any changes causing a reduction in fitness.

To address the chromosomal growth displayed by GE, TAGE is compared to GE where the selection of crossover points is restricted to the expressed region of the chromosome. This restriction has no effect on the performance of TAGE.

Experiments to examine the following null hypothesis are presented:

H_{c0} : There is no difference in performance between TAGE and GE when selecting crossover points from the encoding region exclusively

with the accompanying alternate hypothesis:

H_{c1} : There is a difference in performance between TAGE and GE when selecting crossover points from the encoding region exclusively

The comparison between TAGE and standard GE from Section 6.2.1 is repeated with crossover points only being chosen from the encoding region of GE chromosomes. The best and mean population fitness values are recorded. H_{c0} is tested by comparing the distributions of best fitness values produced by both representations at the final generation of each run using the Mann-Whitney U, two-tailed, non-parametric test with a 5% level of significance.

Results

The effect of this crossover restriction on the growth of the GE chromosome is large and is shown for the quartic symbolic regression problem in Figure 6.11, with a reduction from a peak mean chromosome length of ~600 codons to ~200 codons for the initial chromosome length of 120 codons. The effect of this restriction on performance is also quite large, with improvements for GE in both the mean best fitness values and the total number of successful runs across all problems. These values are reported in Table 6.7.

The results of the two-tailed Mann-Whitney U significance tests on the distributions of best fitness values at the final generation are shown in Table 6.6. The improvement seen in GE reduces the number of comparisons between TAGE and GE where the distributions of values differ with p-values < 0.05 to seven out of the 16 comparisons performed on the even parity problem. With GE having better mean best fitness values than TAGE for five of those seven occurrences, specifically when TAGE is initialised with chromosome lengths of 60 and 120. Similarly, for the Santa Fe ant trail problem, TAGE does best when initialised with shorter chromosome lengths, while conversely, GE does best when it is initialised with the larger chromosome lengths.

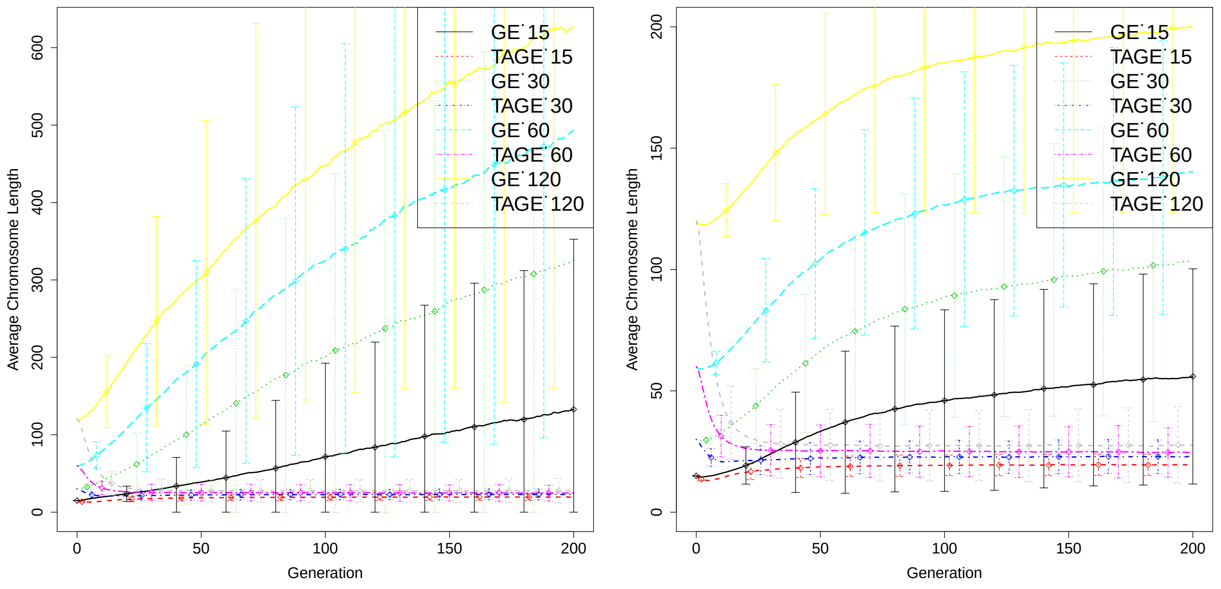


Figure 6.11: Mean chromosome length with no restriction on crossover points (left) and with only encoding region crossover points (right) plots for GE and TAGE for the quartic symbolic regression problem with error bars of one standard deviation. The mean best fitness and mean chromosome length plots for all problems are provided in Figure B.14 and Figure B.15 respectively.

Table 6.7: A comparison of results obtained by GE and TAGE across the benchmark problems with encoding region restricted crossover. The mean best fitness values across 500 runs for the final generation and the total number of successful runs are reported, correct to four decimal places.

		GE		TAGE	
		Mean Best Fitness (SD)	Successes	Mean Best Fitness (SD)	Successes
EFP	15	0.144 (0.9278)	486	0.032 (0.3567)	496
	30	0.04 (0.4180)	495	0.058 (0.5051)	491
	60	0.028 (0.2956)	495	0.076 (0.4719)	484
	120	0.066 (0.4405)	487	0.144 (0.7101)	475
SF	15	18.472 (12.0756)	59	9.13 (10.6576)	224
	30	12.996 (12.5883)	141	12.246 (12.0733)	189
	60	9.46 (11.3433)	177	14.896 (12.6748)	137
	120	9.968 (11.7367)	204	17.776 (13.7793)	113
SR	15	0.1148 (0.1227)	249	0.0007 (0.0057)	489
	30	0.0354 (0.0845)	414	0.0016 (0.0091)	480
	60	0.0079 (0.0401)	470	0.0017 (0.0083)	476
	120	0.0069 (0.0366)	474	0.0034 (0.0154)	467
6 Mux	15	4.154 (3.9198)	175	0.71 (1.9103)	423
	30	4.008 (3.7202)	173	0.742 (1.7815)	411
	60	4.128 (3.7389)	159	0.802 (1.6493)	381
	120	4.552 (4.6941)	133	1.312 (2.2461)	337

Table 6.8: Mann-Whitney U test results for best fitness values at generation 200 with varied initial chromosome size and encoding region restricted crossover. Shaded cells represent p-values < 0.05. p-values are provided in Table B.4. Each cell is labelled with T or G, indicating whether TAGE or GE has a better mean best fitness. - indicates the same value for both TAGE and GE.

		GE															
		EFP				SF				SR				6 Mux			
		15	30	60	120	15	30	60	120	15	30	60	120	15	30	60	120
TAGE	15	T ▨	T	G	T ▨	T ▨	T ▨	T ▨	T	T ▨	T ▨	T ▨	T ▨	T ▨	T ▨	T ▨	T ▨
	30	T	G	G	T	T ▨	T ▨	G ▨	G ▨	T ▨	T ▨	T ▨	T ▨	T ▨	T ▨	T ▨	T ▨
	60	T	G ▨	G ▨	G	T ▨	G	G ▨	G ▨	T ▨	T	T	T	T ▨	T ▨	T ▨	T ▨
	120	-	G ▨	G ▨	G ▨	T ▨	G ▨	G ▨	G ▨	T ▨	T	T ▨	T ▨	T ▨	T ▨	T ▨	T ▨

6.2.4 Discussion

From the differences in the representations examined in this section it is seen that initial chromosome length plays an important role in the performance of both GE and TAGE. When initialised with the similar chromosome lengths using random chromosomes the two representations behave quite differently.

Shorter chromosomes are preferred by TAGE due to the size of TAGE derived trees and resulting phenotypes being directly proportional to the chromosome size. TAGE performs worse when initialised with a population consisting entirely of very large individuals.

Inversely, longer initial chromosomes are desirable for GE. Random initialisation in GE tend towards generating smaller derivation trees [63]. Increasing the initial chromosome length doesn't have a large effect on the size of generated derivation trees. However, by lengthening the chromosome, a larger non-coding region, or tail, is provided. Longer tails increase the chance that individuals will terminate while mapping from genotype to phenotype [157]. Therefore, increasing the length of the initial chromosomes not only reduces the number of invalids in the initial population, but also throughout the run thereafter. Invalid individuals are undesirable in the population as they reduce the diversity of the population, in addition to reducing selection pressure by participating in selection tournaments.

Due to the trend of GE generating smaller trees when randomly initialised, other initialisation methods are preferred [63], and are discussed in the context of TAGE in Chapter 8. TAGE, however, does not suffer from the same initialisation problems as GE. As the size of TAGE derivation trees is a function of chromosome length, ramped random initialisation is a viable, computationally inexpensive, initialisation method for generating individuals of varied sizes when using TAGs in GP [61].

TAGE also does not suffer from the generation of invalid individuals. While using longer initial chromosomes in the population reduces the effect of invalids in GE and replacing them entirely throughout a run with parent individuals, TAGE remains to perform significantly better than GE on the majority of setups comparing the problems.

In addition, ensuring that the operators used by GE do not bias their operation on the non-coding tail of GE chromosomes is very important for the performance of GE. When this is enforced, by allowing crossover points to be only chosen with the encoding region, GE operates as well as TAGE in a number of comparisons for three of the four problems examined, and indeed, performing significantly better than TAGE when initialised with longer chromosome lengths, of 60 and 120 codons in particular, on the even five parity and Santa Fe ant trail problems. The operators used by TAGE, can only operate on the expressed region as TAGE has no non-coding tail. A side effect of this is that TAGE, with at most a single intron, has little ability for neutral evolution, whereby chromosome modification doesn't bring about a change in phenotype and/or fitness. The only opportunities for neutral evolution at the phenotypic level in TAGE is that which is provided by the redundant mapping inherent to the GE genotype-phenotype mapping rule, as well as between any symmetric elementary trees that exist in the TAG. There is more opportunity for neutral evolution at the fitness level, with TAGE's specific landscape connectivity (discussed in Chapter 7) potentially allowing access to fitness neutral pathways.

For completeness, the mean best fitness values and total number of successful runs for GE with both population validation and encoding region only crossover points enabled are provided in Table 6.9, with statistical significance reported in Table 6.10.

6.3 Summary

In this chapter a comparison is conducted between TAGE and canonical GE, which demonstrates that the use of TAGs in GE can have a beneficial effect on ability of the GE algorithm to move through the solution search space and to find successful solutions. However,

Table 6.9: A comparison of results obtained by GE and TAGE across the benchmark problems with population validation enabled and encoding region restricted crossover. The mean best fitness values across 500 runs for the final generation and the total number of successful runs are reported, correct to four decimal places.

		GE		TAGE	
		Mean Best Fitness (SD)	Successes	Mean Best Fitness (SD)	Successes
EFP	15	0.194 (1.0151)	480	0.032 (0.3567)	496
	30	0.168 (0.9238)	482	0.058 (0.5051)	491
	60	0.326 (1.1994)	460	0.076 (0.4719)	484
	120	0.362 (1.2674)	454	0.144 (0.7101)	475
SF	15	16.004 (12.7729)	99	9.13 (10.6576)	224
	30	10.946 (11.8881)	161	12.246 (12.0733)	189
	60	10.018 (11.6039)	168	14.896 (12.6748)	137
	120	10.012 (11.5394)	178	17.776 (13.7793)	113
SR	15	0.0688 (0.1094)	312	0.0007 (0.0057)	489
	30	0.0259 (0.0687)	387	0.0016 (0.0091)	480
	60	0.0193 (0.0593)	409	0.0017 (0.0083)	476
	120	0.0138 (0.0470)	414	0.0034 (0.0154)	467
6 Mux	15	4.658 (3.9674)	154	0.71 (1.9103)	423
	30	4.578 (3.7309)	131	0.742 (1.7815)	411
	60	4.478 (3.7823)	141	0.802 (1.6493)	381
	120	4.514 (3.6814)	133	1.312 (2.2461)	337

Table 6.10: Mann-Whitney U test results for best fitness values at generation 200 with varied initial chromosome size, population validation enabled and encoding region restricted crossover. Shaded cells represent p-values < 0.05. p-values are given in Table B.5. Each cell is labelled with T or G, indicating whether TAGE or GE has a better mean best fitness.

		GE															
		EFP				SF				SR				6 Mux			
		15	30	60	120	15	30	60	120	15	30	60	120	15	30	60	120
TAGE	15	T ▨	T ▨	T ▨	T ▨	T ▨	T ▨	T ▨	T ▨	T ▨	T ▨	T ▨	T ▨	T ▨	T ▨	T ▨	T ▨
	30	T ▨	T	T ▨	T ▨	T ▨	G	G	G	T ▨	T	T ▨	T ▨	T ▨	T ▨	T ▨	T ▨
	60	T	T	T ▨	T ▨	T ▨	G	G	G	T ▨	T	T	T	T ▨	T ▨	T ▨	T ▨
	120	T	T	T ▨	T ▨	G	G	G	G	T ▨	T	T	T	T ▨	T ▨	T ▨	T ▨

a number of differences between the representations are highlighted. This leads to further analysis of the two representations.

These representational differences are singled out and examined by performing further comparisons. It is shown that, while each of the differences contribute towards the performance improvement shown by TAGE on a number of problems, further analysis is required to better understand how TAGE more effectively traverses the search and solution spaces, as compared to GE.

The experiments in this chapter uncover some interesting results and properties of TAGE, including effective initialisation using randomly generated chromosomes, minimal extraneous growth of the chromosome, and the increase of connectivity of the search space which might help TAGE maintain diversity.

The following chapter investigates the property of TAGE outlined at the end of Section 6.1.4, that TAGE populations consistently maintain mean fitness values that are much farther away from the optimum than those of GE populations. This apparent decrease in convergence towards the optimum displayed by TAGE populations is considered, and the effect of variation operations on GE and TAGE is compared. In particular, the effect of the integer flip mutation operator on both representations is examined in order to ascertain how the use of TAGs in GE increases the connectivity of the search space and decreases the rate convergence.

Chapter 7

Examining Connectivity in TAGE - Mutation Landscapes

This chapter examines the change in representation from GE to TAGE. In particular this chapter aims to address an observation made in Chapter 6. It states that on average, the mean fitness values of TAGE populations are farther from the optimum than those of GE. The chapter outlines a landscape model which is used to visualise and compare the two representations when mutation alone is acting upon the individuals. It is shown that under single-change mutation, TAGE is much more connected, i.e., any one phenotype can be mutated to many more other phenotypes than GE. It is also shown that for the grammars used, in conjunction with the integer-flip mutation operation, shorter GE phenotypes have a disproportionately high frequency of connections amongst themselves and to a lesser extent with the rest of the phenotypes. This indicates that the CFGs used in this study have a bias towards shorter phenotypes, i.e., in the enumerated set of genotypes, the fewer codons used to map a specific phenotype, the larger the proportion of that set the genotypes which map to that phenotype make up. The work presented in this chapter is based upon the work published by Murphy et al. [142].

Representation is a very important component of any evolutionary algorithm. Changing the representation can cause an algorithm to perform very differently. The effect of such a change may not be immediately obvious and can be difficult to understand.

One such change is outlined in Section 6.1.4. By extending GE to make use of TAGs, rather than the standard CFGs, an increase is observed in the ability of the algorithm to find solutions with better fitness in fewer generations, as well as finding more successful solutions overall for the problems examined (Table 6.2). While the best fitness in the population is better, on average, for TAGE than that of GE, the mean fitness of TAGE populations are further from the optimum than those of GE.

This chapter is concerned with this change in representation, from GE to TAGE. In particular, the chapter aims to better understand how that change produces the observed effect of worse mean population fitness. Previous work has examined how the TAG representation overcomes some of the structural difficulties present in GP [85], but the full extent of how TAGs affect GE is unclear. Koza and Poli [111] said that visualising the program search space would be useful and help us understand how an algorithm operates. However, viewing the entire search space is difficult due to its large size and high complexity.

Landscapes, a tool to help understand complex processes [95], are employed here in an attempt to further understand the GE algorithm and how the choice of representation affects its performance. Using a single variation operation, Integer Flip Mutation (IFM), landscapes for both TAGE and GE are examined. The IFM operation, as described in Section 2.4.4, is the modification of a single codon, replacing the existing value with a new integer value drawn from a uniform distribution.

This study compares visualisations of the single IFM landscapes of both GE and TAGE on the grammars of a series of problems in an attempt to further understand how the change in representation affects the algorithm's ability to search. To alleviate the visualisation problem, subsets of the landscapes are visualised by means of a method little used in the field of GP, adjacency matrices.

The remainder of this chapter is organised as follows. Section 7.1 provides an introduction to the landscape model used in this study. Section 7.2 outlines the generation of homologous landscapes for both representations, ensuring that the landscapes are comparable. The methods of visualisation and the resulting adjacency matrices are presented in Section 7.3, followed by a discussion of these results in Section 7.4. Conclusions and a summary of the chapter are given in Section 7.5.

7.1 Landscapes

The landscape model used in this paper is as defined by Jones [95], where he quotes Nilsson [160], *"In its broadest sense, problem solving encompasses all of computer science because any computational task can be regarded as a problem to be solved."*, Pearl [185], *"Every problem-solving activity can be regarded as the task of finding or constructing an object with given characteristics"*, and Rich [194], *"Every search process can be viewed as a traversal of a directed graph in which each node represents a problem state and each arc represents a relationship between the states represented by the nodes it connects"*, stating that from the above statements one can conclude that search is ubiquitous and that it can be described as a process on a graph structure [95]. It is for this reason that he adopts a graph as a view of his landscape model.

The landscape model, L, is written as the quintuple

$$L = (R, \phi, f, F, >_F) \tag{7.1}$$

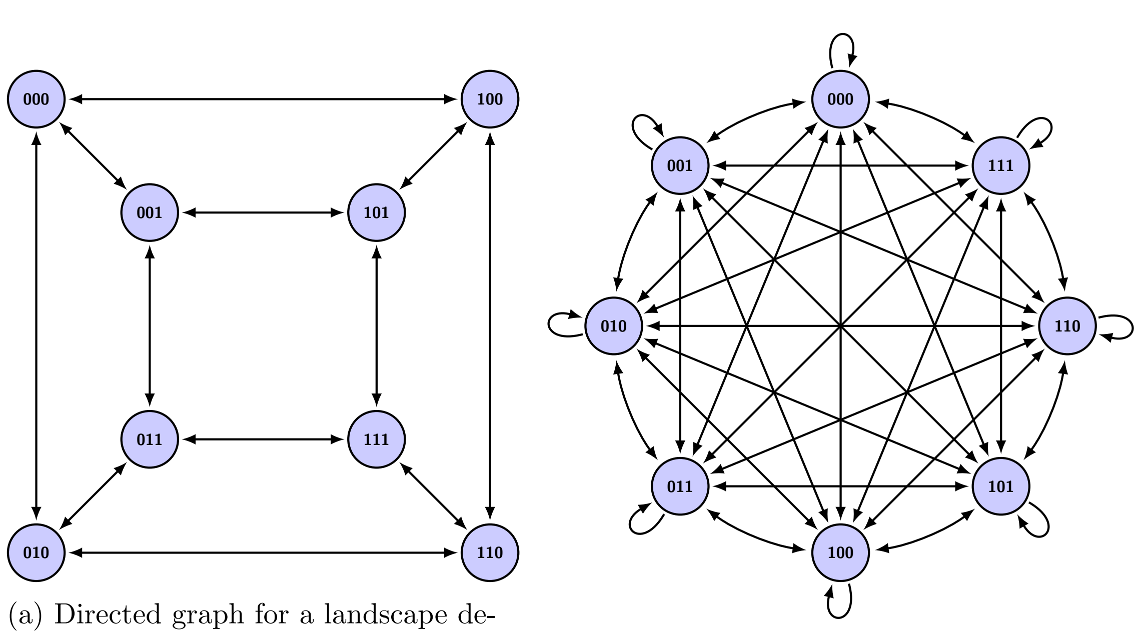
where R is the representation space, ϕ is the operator (in this case a genetic operator), the function f which maps a multi-set of R to the fitness space, F - i.e., $f : M(R) \mapsto F$, or f is a many-to-one mapping of R onto F - and a partial ordering $>_F$ over F.

In order to make use of the landscape model to gain some insight into the behaviour of the system it embodies, L can be used to define a directed labelled graph, $G_L = (V, E)$

where $V \subseteq M(R)$, $E \subseteq V \times V$ and $(v, w) \in E \iff \phi\,(v, w) > 0$. This notation, $\phi\,(v, w)$, is introduced by Jones [95] and is used to represent the probability of the operator, ϕ, when applied to v, resulting in w, i.e. $P\,(\phi\,(v) \mapsto w)$. If this probability is zero, then no such edge exists in the landscape. Each vertex, v, is labelled with $f\,(v)$.

Figure 7.1 presents two example landscape models as directed graphs where R is the set of all binary strings of length three. The first model (Figure 7.1a) is a single-change operator class landscape. This class of operator, which is used by this study, acts upon and results in a single element of R. A single component of the input is selected and modified, producing an output at exactly a distance of one from the input. If the elements of R have n binary components then the graph produced is a hyper-cube of n dimensions (each vertex has a neighbourhood of size n), where each vertex can have a maximum degree of n, and each edge in the graph has a minimum transition probability of $1/n$. Single-change operators are the simplest class of operator defined by Jones [95].

The second model (Figure 7.1b), which is not used in this study but rather as a counter point, is a mutational operator class landscape. This landscape highlights how complex landscapes can become even while only acting upon R. Rather than acting on a single component of their input, every component of the input is modified according to some probability distribution. The probability of changing any single component is typically small, however, as every component has the ability to change, the distance of the produced output from the input can vary from zero, to completely different. Jones [95] goes on to define two further operator classes, crossover operators which act on R^2, combining components of pairs of inputs, producing pairs of outputs; and selection operators which act on entire populations ($M\,(R)$). This thesis does not make use of these operators and directs the reader to Jones [95] for further details.



(a) Directed graph for a landscape defined by a single-change operator.

(b) Directed graph for a landscape defined by a mutational operator.

Figure 7.1: Example landscapes for two different operators classes. R, the set of all length three binary strings and ϕ, the bit flip mutation operation. Edge probabilities are omitted.

7.2 Landscape Design and Generation

This section outlines how to make use of the landscape model to provide insight into the behaviour of GE and TAGE when applying single-change IFM operations to their genotypes. The landscapes are defined using the space of valid genotypes, or chromosomes, as R. A valid genotype is one that successfully maps to a phenotype. ϕ is a single-change class operator, the single IFM operator. In this study f is left unset, as the landscapes are being examined for connectivity alone.

As both GE and TAGE share the same phenotypic space, the produced landscapes are projected into this space to allow for a comparison between them. Figure 7.2 portrays the many-to-one projection of the landscape from genotypic to phenotypic spaces. These landscapes can be viewed as graph structures either where $V \subseteq M(R)$, the representation (genotype) space as described above, or where $V \subseteq g(R)$ and g maps from genotype to

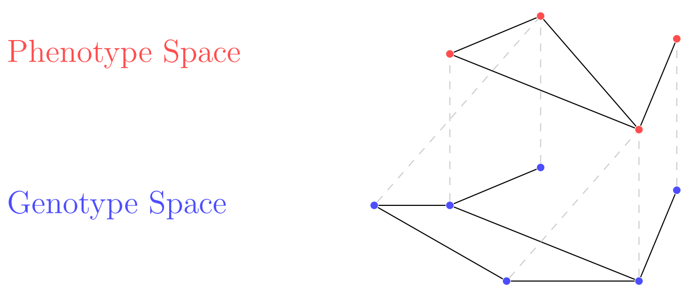


Figure 7.2: Genotype landscape projected into the phenotypic space.

phenotype.

For both types of graph , the property that $E \subseteq M(R) \times M(R)$ is retained, i.e., the neighbourhood of a vertex is dependent on ϕ and R (neighbouring genotypes). The graph model in which V consists of phenotypes can be though of as being the original R-based graph, in which the nodes whose genotypes map to matching phenotypes are collapsed down to a single node.

7.2.1 Restricting Landscape Complexity

In Section 7.1 the landscape model is described and two basic examples are provided in Figure 7.1. While these example landscapes can be easily graphed, as the string length, and hence the number of nodes in the landscape increases, as will happen when applied to GE chromosomes, the complexity of these graphs will also increase. This makes visualisation much more difficult. Additionally, each codon in the chromosome can be used to choose between two or more productions per rule, as opposed to the binary examples provided. This results in a further increase of the complexity of the graphs. In order for visualisation, the landscape must be restricted in some manner.

As the complexity of the landscapes cannot be restricted in terms of the degree of each node, due to this being a property defined by the grammar, the chromosome length is restricted to allow for visualisation. A specific number of codons, N, is selected as the maximum length of a chromosome. A value for N is independently chosen for each grammar examined in order to sufficiently restrict the size of the landscapes making them feasible to work with.

At each chromosome length, from length one to N, an enumeration of all possible chromosomes is performed, building up the representation space, R. It is required that each successive length of chromosome is processed, as with TAGE, mutation alone is unable to modify the number of codons used when mapping, and as such, TAGE would not be able to represent the same set of phenotypes as GE.

7.2.2 Enumeration of the Genotype Space

The enumeration process below is performed for TAGE, producing a set of edges between neighbouring genotypes in the landscape, as well as a set of phenotypes. The resulting phenotypes are used to repeat the process for GE. Rather than setting a chromosome length limit, each successive chromosome length is processed until the set of phenotypes generated by GE contains the set of phenotypes generated by TAGE. The landscapes are generated from the intersection of the two sets of generated phenotypes. Specifically from the genotypes which map to generate these phenotypes and the connections between them. An outline of the algorithm is provided in Algorithm 7.1.

Chromosome Enumeration

To begin the enumeration, an empty queue of chromosomes is created and a single initial chromosome of length n is added to the queue. The codons of this initial chromosome are all assigned the same value of zero. A chromosome, i, is removed from the queue, and beginning from the left most codon, i is mapped in a step-wise manner into a phenotype. Upon reading a codon during this mapping process, the number of mapping choices, c, is noted. c represents the number of possible IFM operations that could be performed to i at the current codon. This ignores redundant mutations, i.e., mutating codons to values $\geq c$, as these would produce equivalent chromosome as the mutations $< c$ when using the GE mapping function. Each of these c mutation operations are independently applied to i, producing c new chromosomes. Each of these neighbouring chromosomes are mapped, and if both i and the neighbour are valid, i.e., if the chromosome maps to an executable solution (for TAGE this is not an issue, since all chromosomes are valid), an edge/connection is recorded between i and this chromosome. If any of the newly generated chromosomes have never been observed, they are added to the queue. The mapping of the unmodified i continues and the next codon is read, counting the number of possible mapping choices, generating mutated chromosomes, recording edges, and adding the new chromosomes to the queue if required. This process continues until the mapping process for i terminates, a new chromosome is then removed from the front of the queue and the process repeats itself until the queue is empty.

Once this queue of chromosomes is depleted, the value of n is incremented by one and entire process is repeated with a new initial chromosome, only halting once $n > N$ and the all chromosomes of lengths, up to and including N, have been processed. An outline of the algorithm is provided in Algorithm 7.1.

7.2.3 Grammars Examined

The landscapes generated by standard GE are compared to those generated by TAGE using grammars commonly used for four classic benchmark problems taken from the GP literature: even five parity, quartic symbolic regression, six multiplexer and the max problem. The grammars which generate the GE landscapes are shown in Figure 7.3. These grammars are transformed into TAGs using Algorithm 3.1, which are then used to generate the TAGE landscapes. The generated TAGs are provided in Appendix B.

Algorithm 7.1 Genotype space enumeration for the generation of the IFM landscape. Genotypes are enumerated from length one up to a maximum length N. The set of edges between valid genotypes of one IFM operation in length is returned.

```
Require: $N$ - a maximum chromosome length
  $edges \leftarrow \emptyset$
  $q \leftarrow [\,]$
  $n \leftarrow 1$
  for $n \leftarrow 1$ to $N + 1$ do
    $init \leftarrow$ generateChromosomeOfZeros($n$)
    $q$.enqueue($init$)
    while ! $q$.isEmpty do
      $i \leftarrow q$.dequeue
      $valid \leftarrow i$.map
      for all $c$ in $i$.codons do
        $choices \leftarrow i$.getMappingChoices($c$)
        for $choice \leftarrow 1$ to $choice > choices$ do
          $j \leftarrow i$.generateMutant($c$, $choice$)
          $jValid \leftarrow j$.map
          if $valid$ and $jValid$ then
            $edges$.add($i$, $j$)
          $q$.add($j$)
  return $edges$
```

7.3 Landscape Visualisation

Viewing the landscapes as 2D graphs is not feasible due to their large size and high complexity. Adjacency matrices are used instead, visualised using heat maps to map the connections in the landscape, eliminating the visual clutter produced by rendering each edge. Heat maps are little used in GP literature and are a compact and effective way of graphically representing data in a two dimensional map, where the values of the variable being visualised are represented as colours. In this case they can be thought of as a visualisation of the adjacency matrix belonging to the graph representation of the landscape.

Rather than using the genotypic landscape, i.e., where each vertex represents a single genotype from the representation, the phenotypic landscape is used. This is done as comparing the genotypes of two different representations may not be useful, whereas both

```
<prog>   ::= <expr>
<expr>   ::= <expr> <op> <expr>
           | ( <expr> <op> <expr> )
           | <var>
           | <pre-op> ( <var> )
<pre-op> ::= not
<op>     ::= "|" | & | ^
<var>    ::= d0 | d1 | d2 | d3 | d4
```

(a) Even Five Parity

```
<expr> ::= ( <op> <expr> <expr> ) | <var>
<op>   ::= + | - | *
<var>  ::= x0 | 1.0
```

(b) Symbolic Regression

```
<B> ::= (<B>) &&(<B>) | (<B>) "||"(<B>)
      | !(<B>)
      | (<B>) ? (<B>) : (<B>)
      | a0 | a1 | d0| d1 | d2 | d3
```

(c) Six Multiplexer

```
<prog> ::= <expr>
<expr> ::= <op> <expr> <expr> | <var>
<op>   ::= + | *
<var>  ::= 0.5
```

(d) Max

Figure 7.3: CFGs in Backus-Naur form used to generate each landscape. The generated TAGs are provided in Appendix B.

representations in question generate the same phenotypic space. Two different types of heat map are used and are outlined in the following sections.

7.3.1 Connection Maps

Connection Maps (CM) are binary heat maps of the adjacency matrix where each cell is either set (black) or not set (white). The set of commonly generated phenotypes label each of the axes. For any pair of phenotypes, if there is an edge in the landscape between any two genotypes which map to those phenotypes, then the shared cell is marked. CMs give insight into how well connected each phenotype is within the landscape. The denser the CM, the greater the representation's ability to move from one phenotype to another.

The CMs for both GE and TAGE with each of the grammars can be seen in Figure 7.4-7.11. The axes of these figures are labelled with the phenotypes sorted in ascending order of length, from the top left to the bottom right.

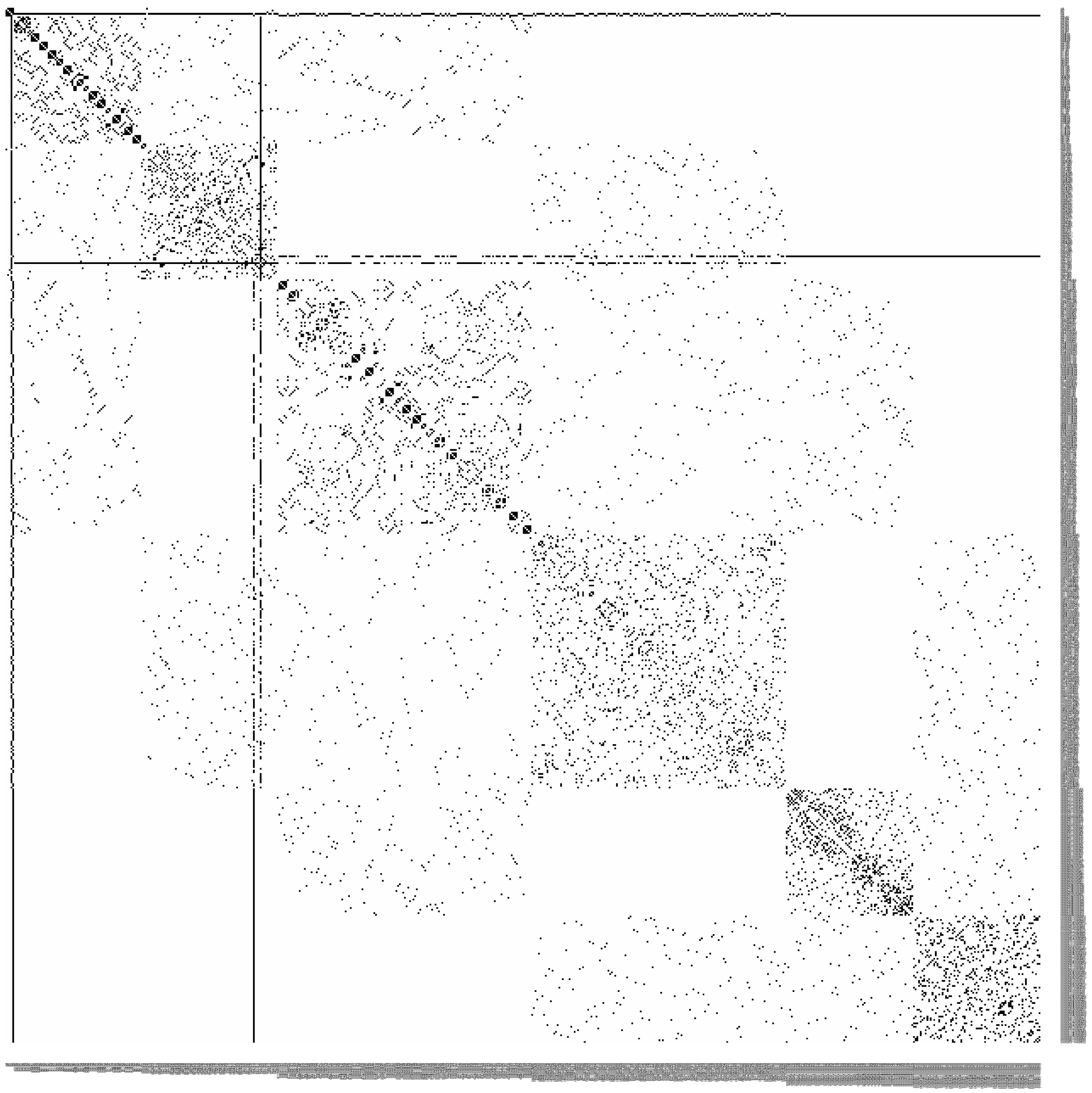


Figure 7.4: GE - Even Five Parity connection maps created with a maximum TAGE chromosome length of three.

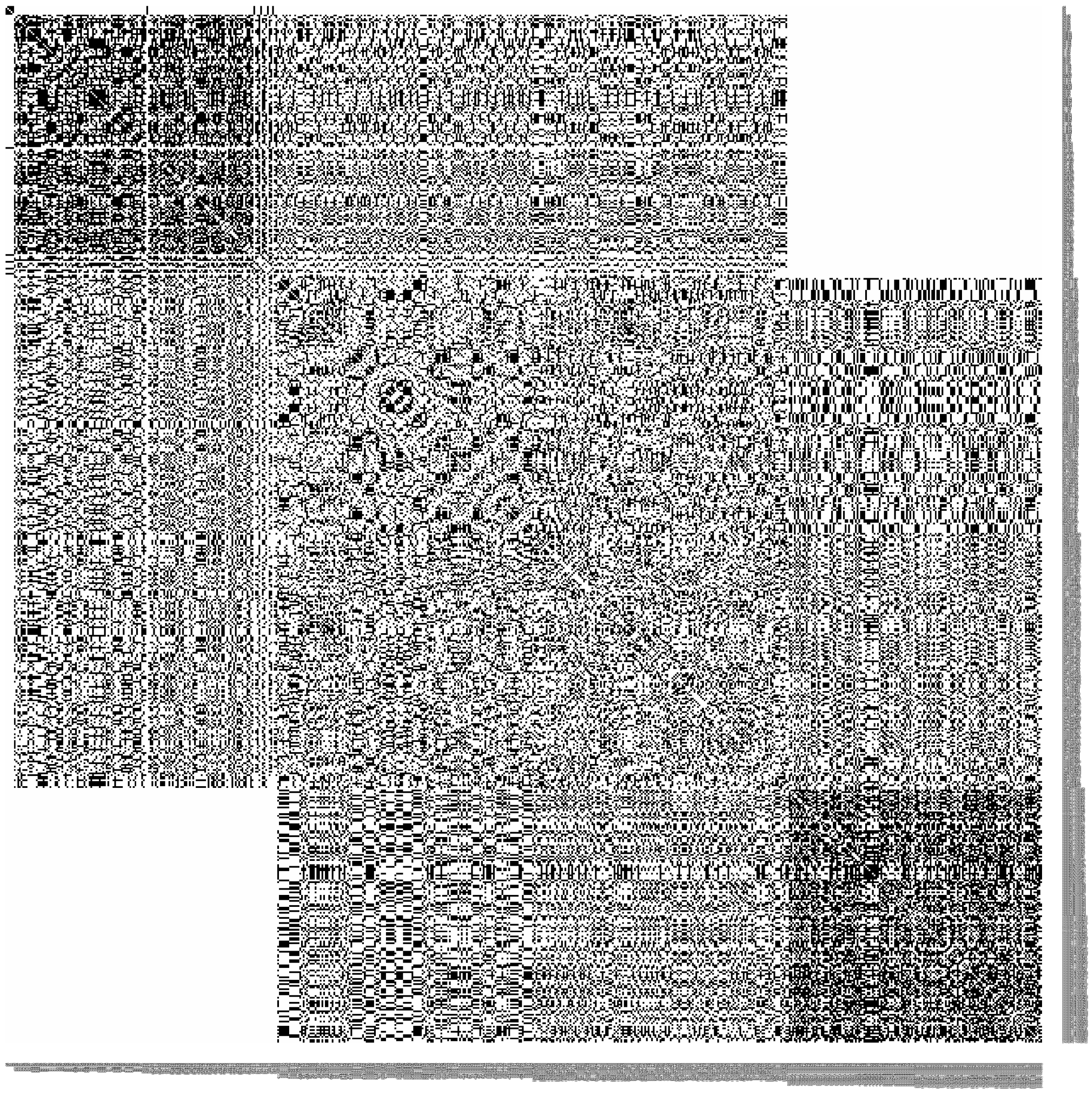


Figure 7.5: TAGE - Even Five Parity connection maps created with a maximum TAGE chromosome length of three.

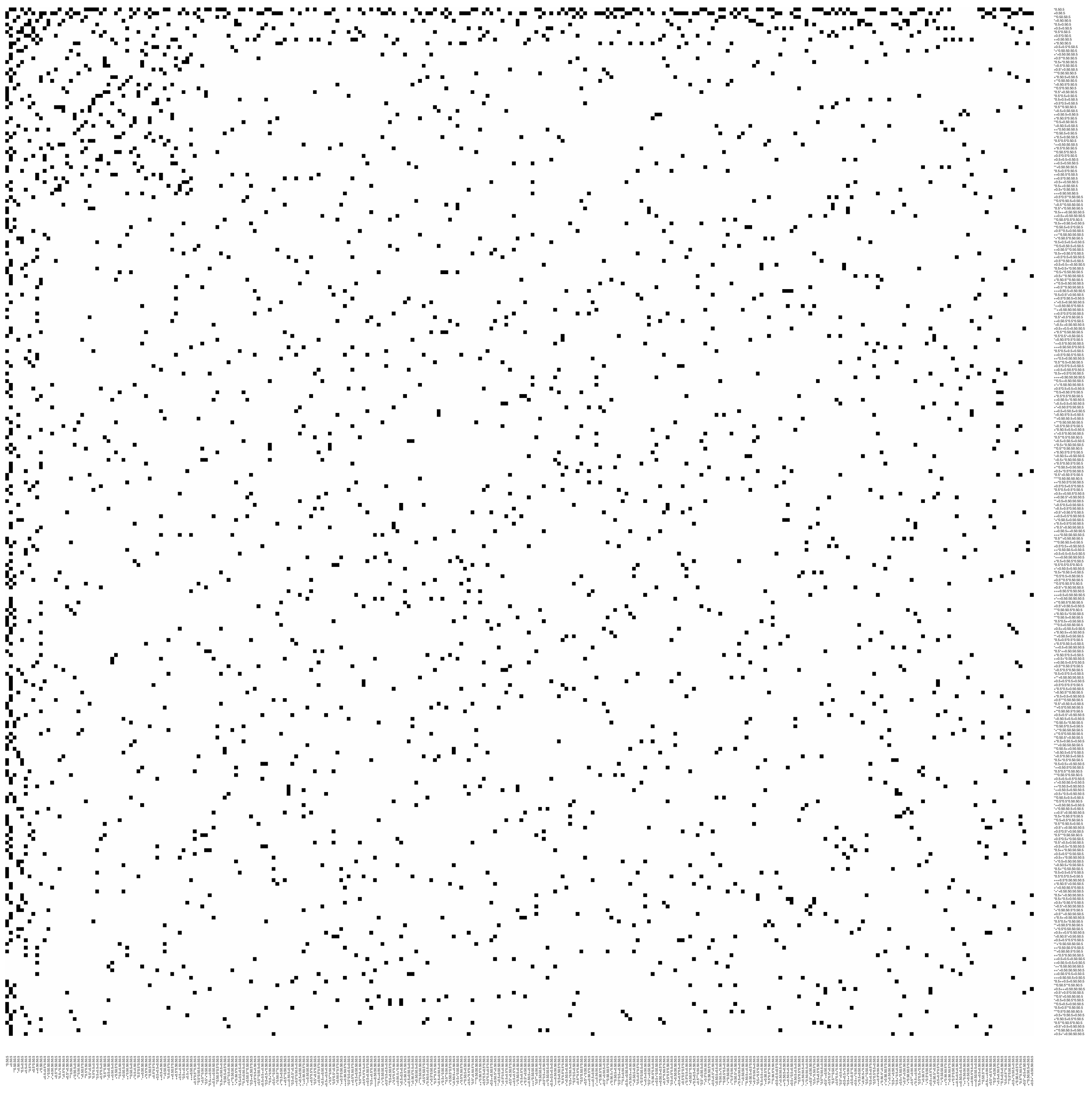


Figure 7.6: GE - Max connection maps created with a maximum TAGE chromosome length of nine.

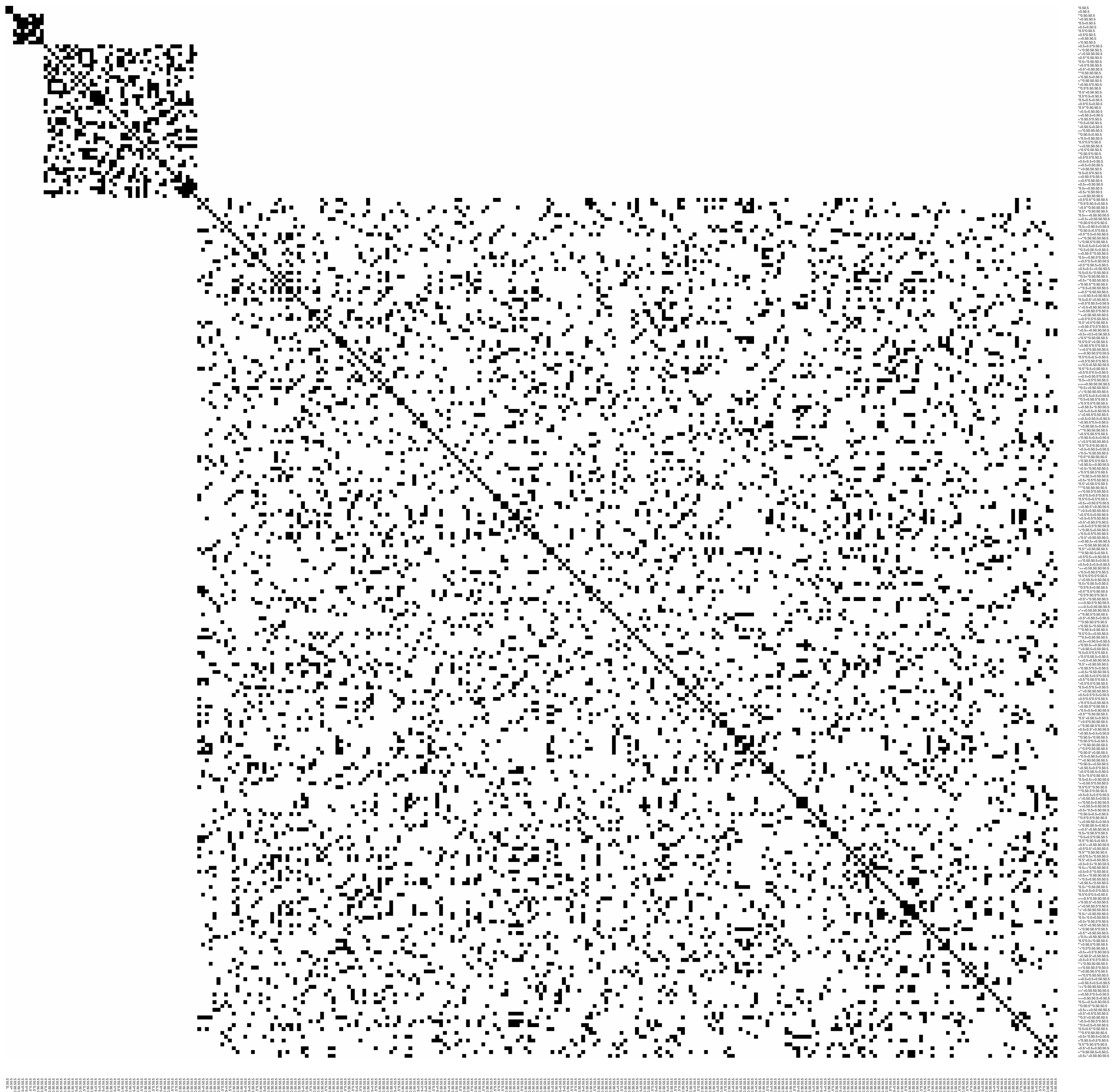


Figure 7.7: TAGE - Max connection maps created with a maximum TAGE chromosome length of nine.

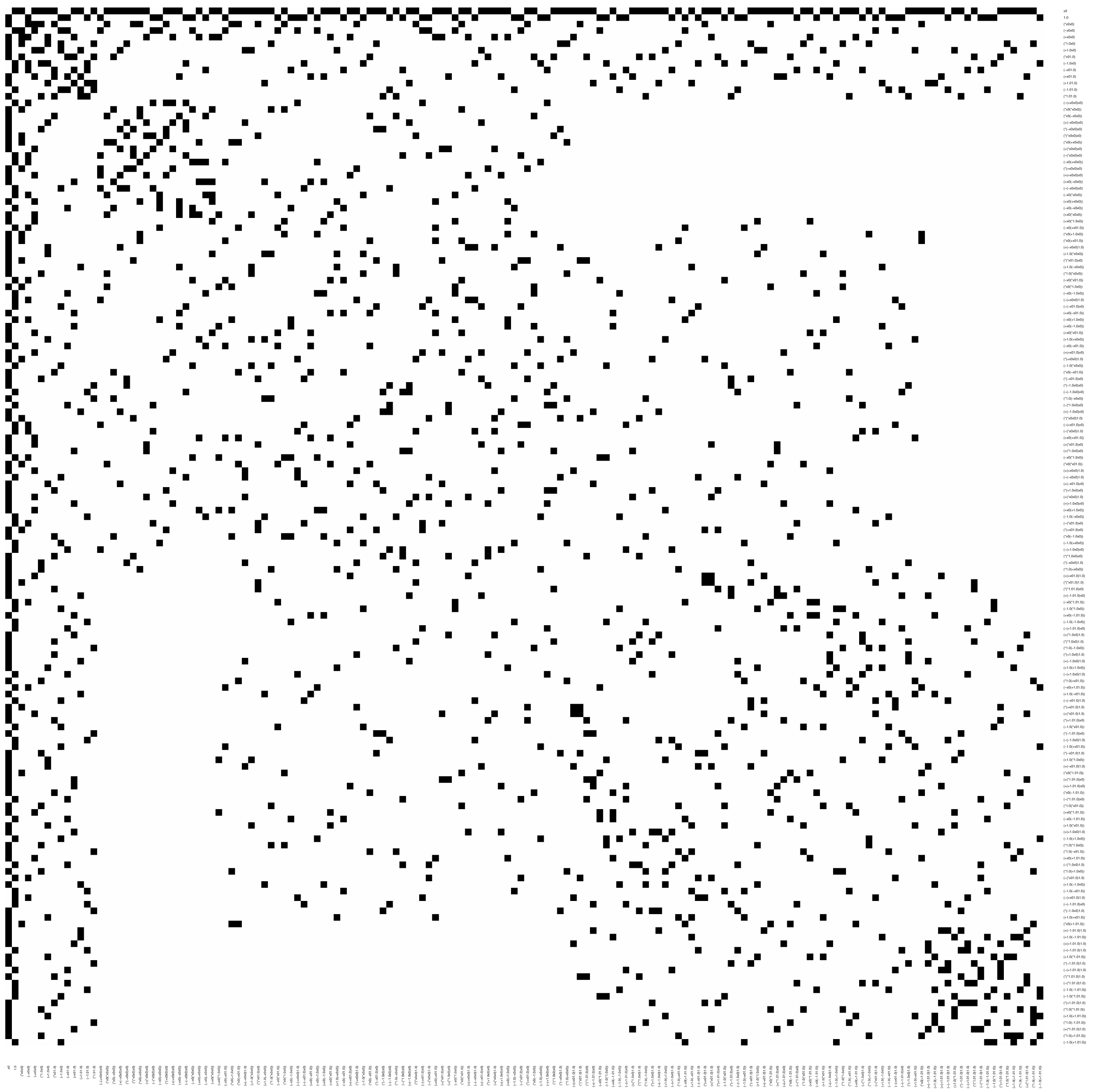


Figure 7.8: GE - Symbolic Regression connection maps created with a maximum TAGE chromosome length of five.

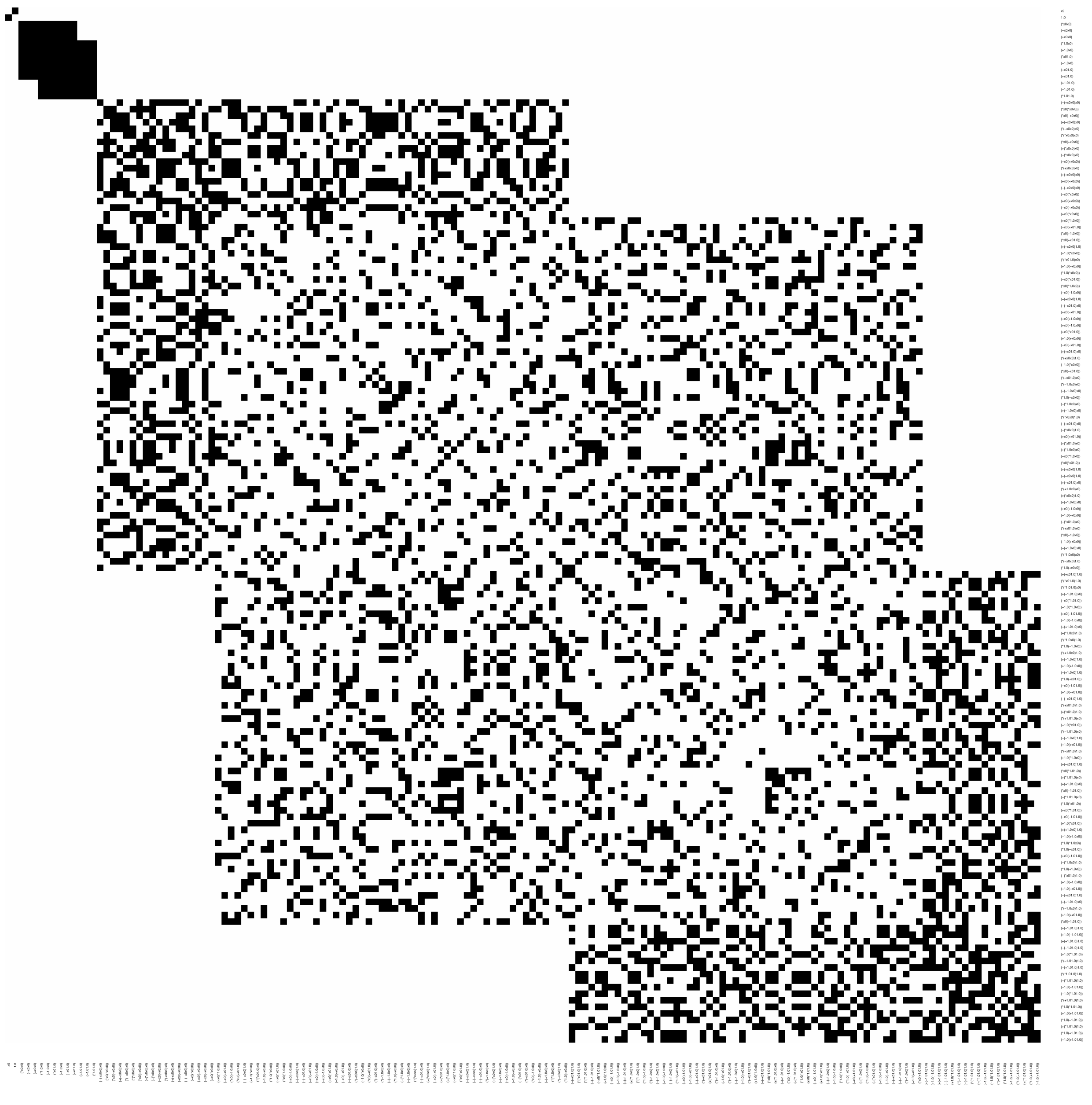


Figure 7.9: TAGE - Symbolic Regression connection maps created with a maximum TAGE chromosome length of five.

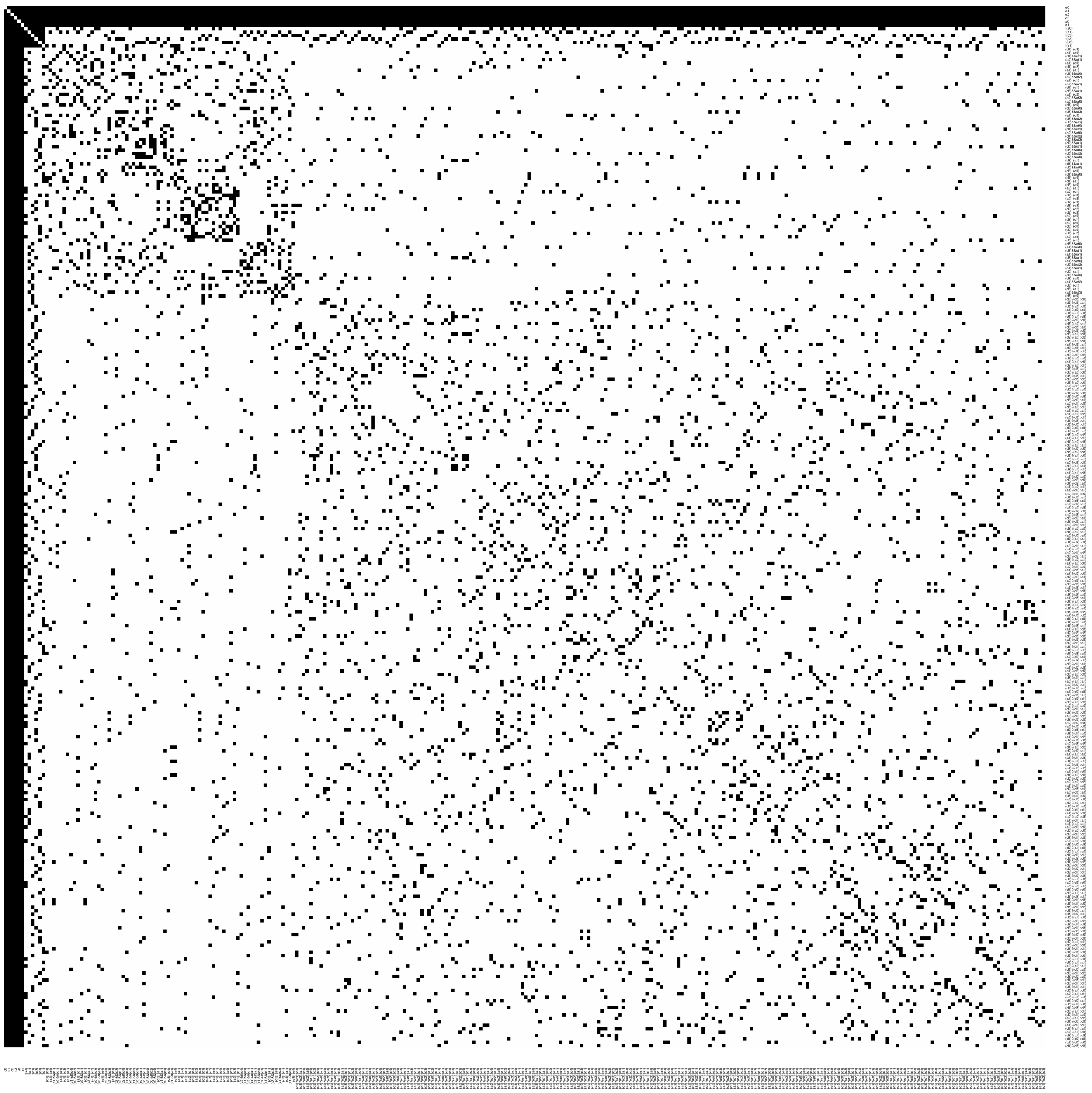


Figure 7.10: GE - Six Multiplexer connection maps created with a maximum TAGE chromosome length of three.

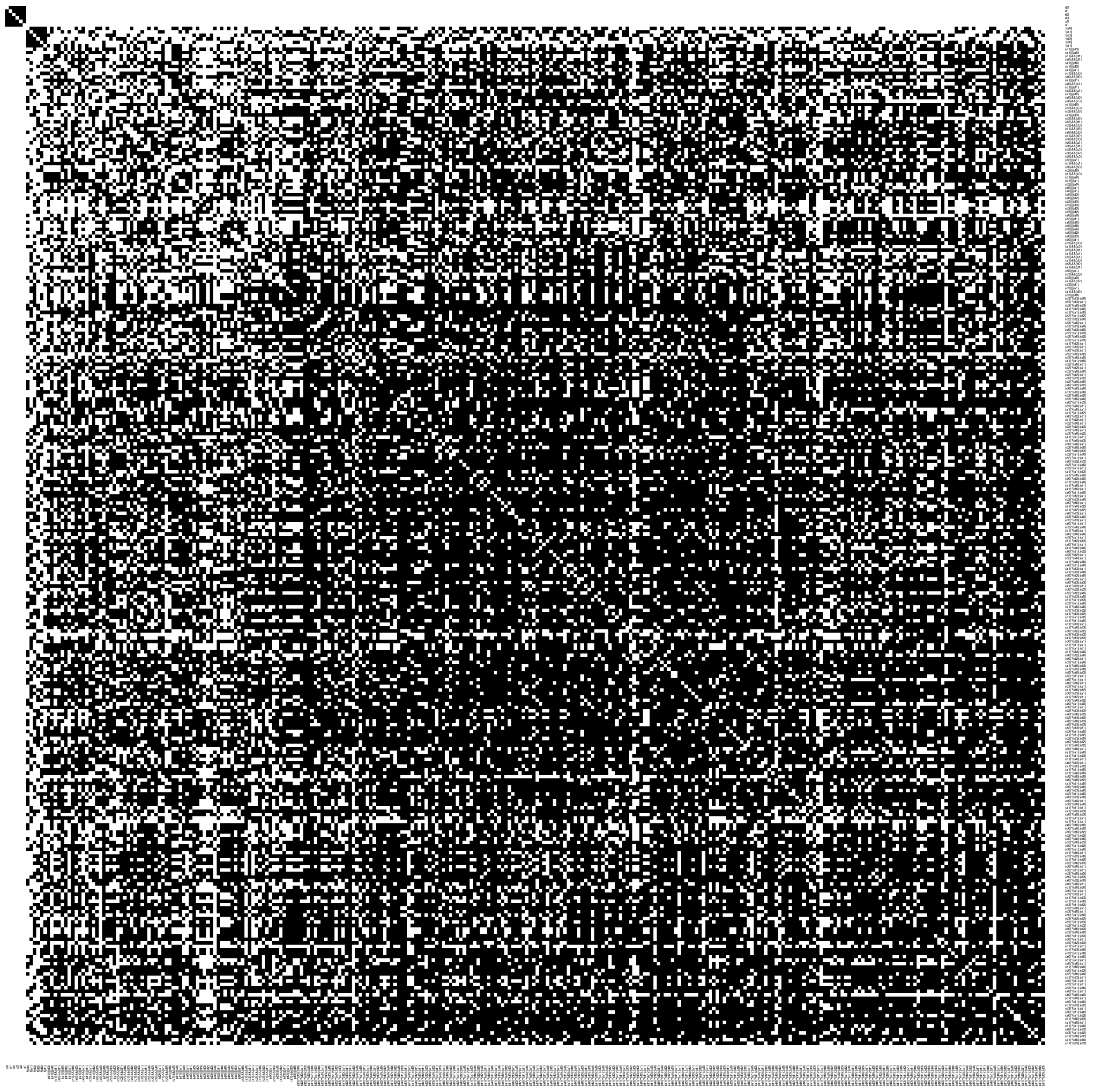


Figure 7.11: TAGE - Six Multiplexer connection maps created with a maximum TAGE chromosome length of three.

7.3.2 Frequency Maps

Frequency Maps address a deficiency of the CMs described above. CMs do not take into account that there may be more than one connection between two phenotypes. This can occur as GE and TAGE have redundant mappings. While redundant codon values were excluded while enumerating the genotype space and generating the landscapes, chromosomes which use many codons to map to a phenotype can be mutated into chromosomes which use fewer codons to map, producing shorter individuals. The frequency of connections between phenotypes is important since if one connection from a phenotype has a high frequency and all of the other connections from that phenotype have a relatively low frequency of connections then there is a much higher probability that a mutation will follow the connections of high frequency. Frequency maps colour each cell from 25% grey (0 connections) to red (200+) depending on the cell's degree of connectivity. The upper bound of 200 connections was to ensure a feasible colour delta when colour coding the maps due to the large number of relatively low frequency cells and a small number of much higher frequency cells. Frequency maps for each of the four grammars on both representations can be seen in Figure 7.12-7.19.

7.4 Discussion

The CMs in Figure 7.4 through to Figure 7.11 show that phenotypes in the TAGE landscapes, across the grammars examined, are much more connected than the same phenotypes in the GE landscapes. This does not necessarily improve TAGE's ability to move from one phenotype to another of better fitness since the concept of fitness is not present in the CMs. It was, however, noted in Section 6.1.4 that TAGE maintains a much larger fitness variance within the population than GE, and that this variance might be a result of a more diverse or disperse population. Diversity can help prevent premature convergence

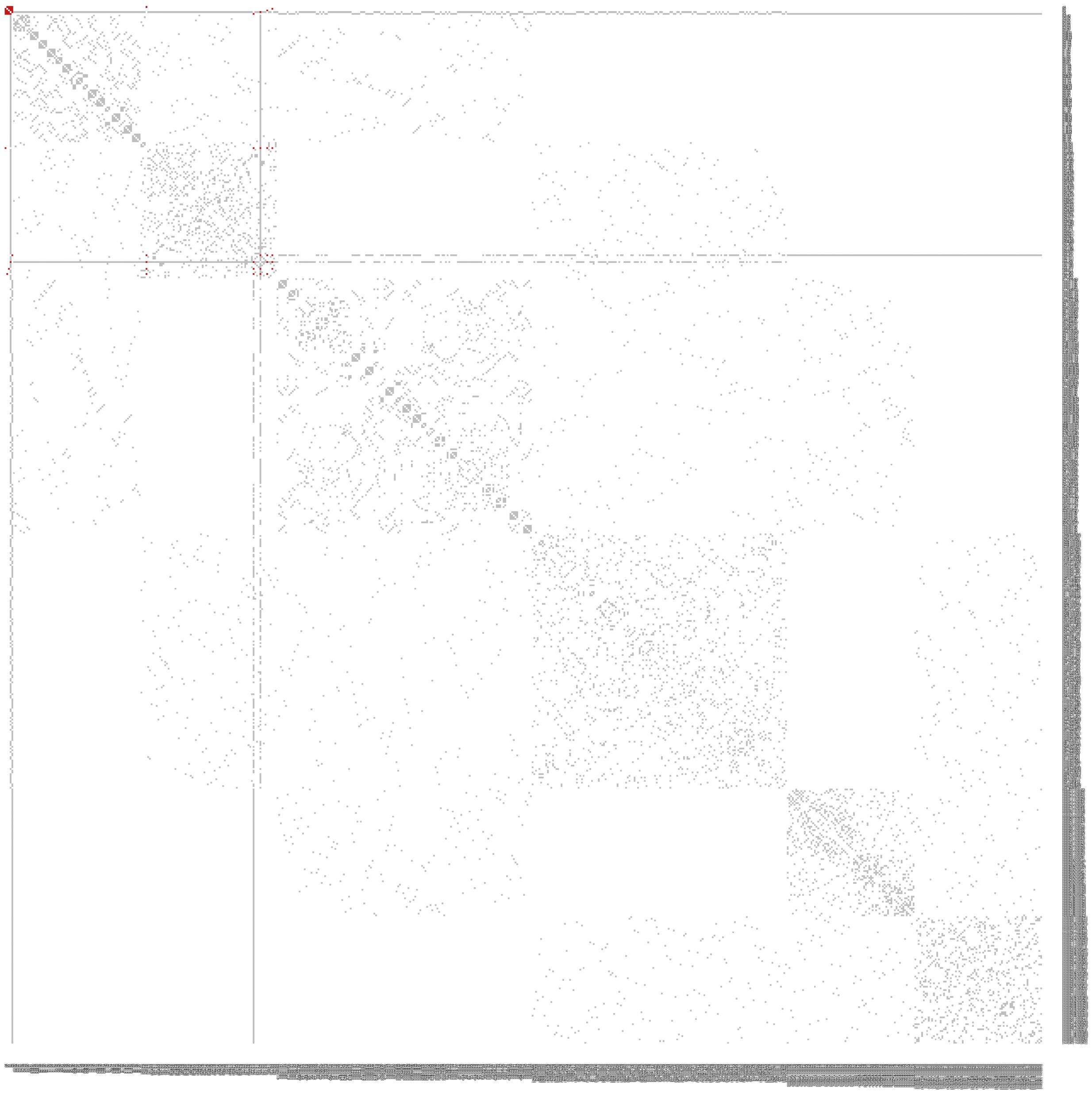


Figure 7.12: GE - Even Five Parity frequency maps created with a maximum TAGE chromosome length of three.

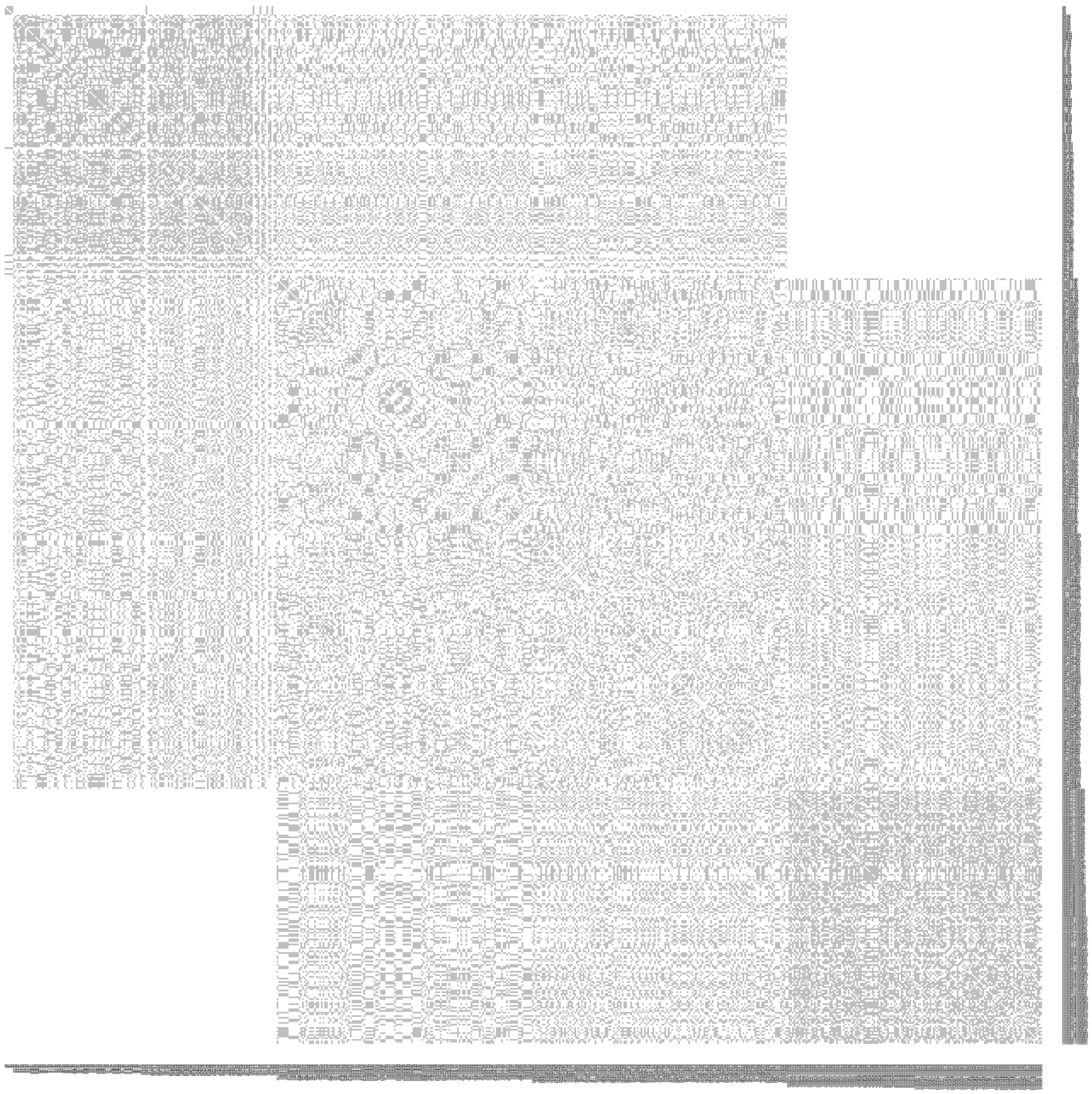


Figure 7.13: TAGE - Even Five Parity frequency maps created with a maximum TAGE chromosome length of three.

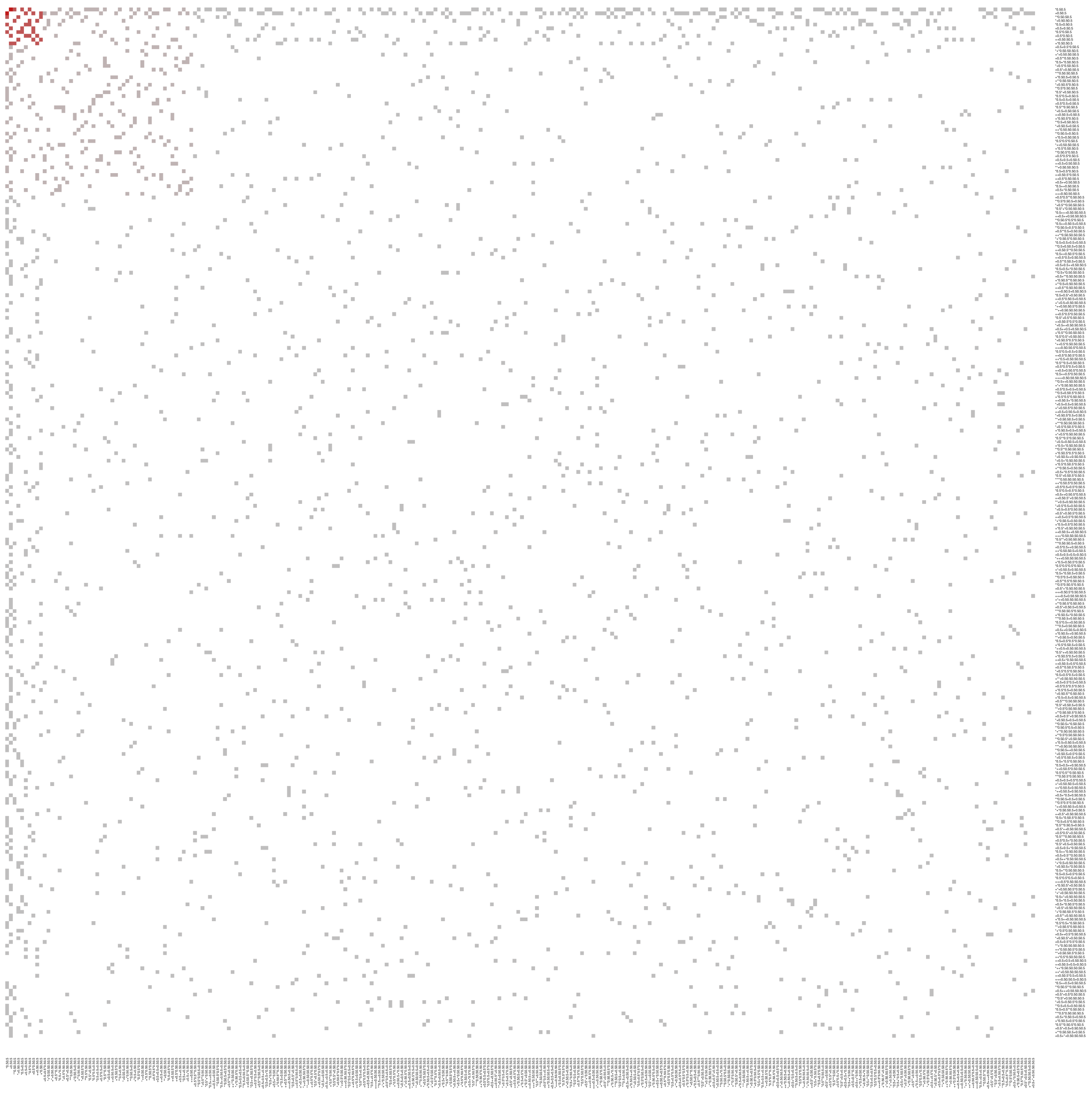


Figure 7.14: GE - Max frequency maps created with a maximum TAGE chromosome length of nine.

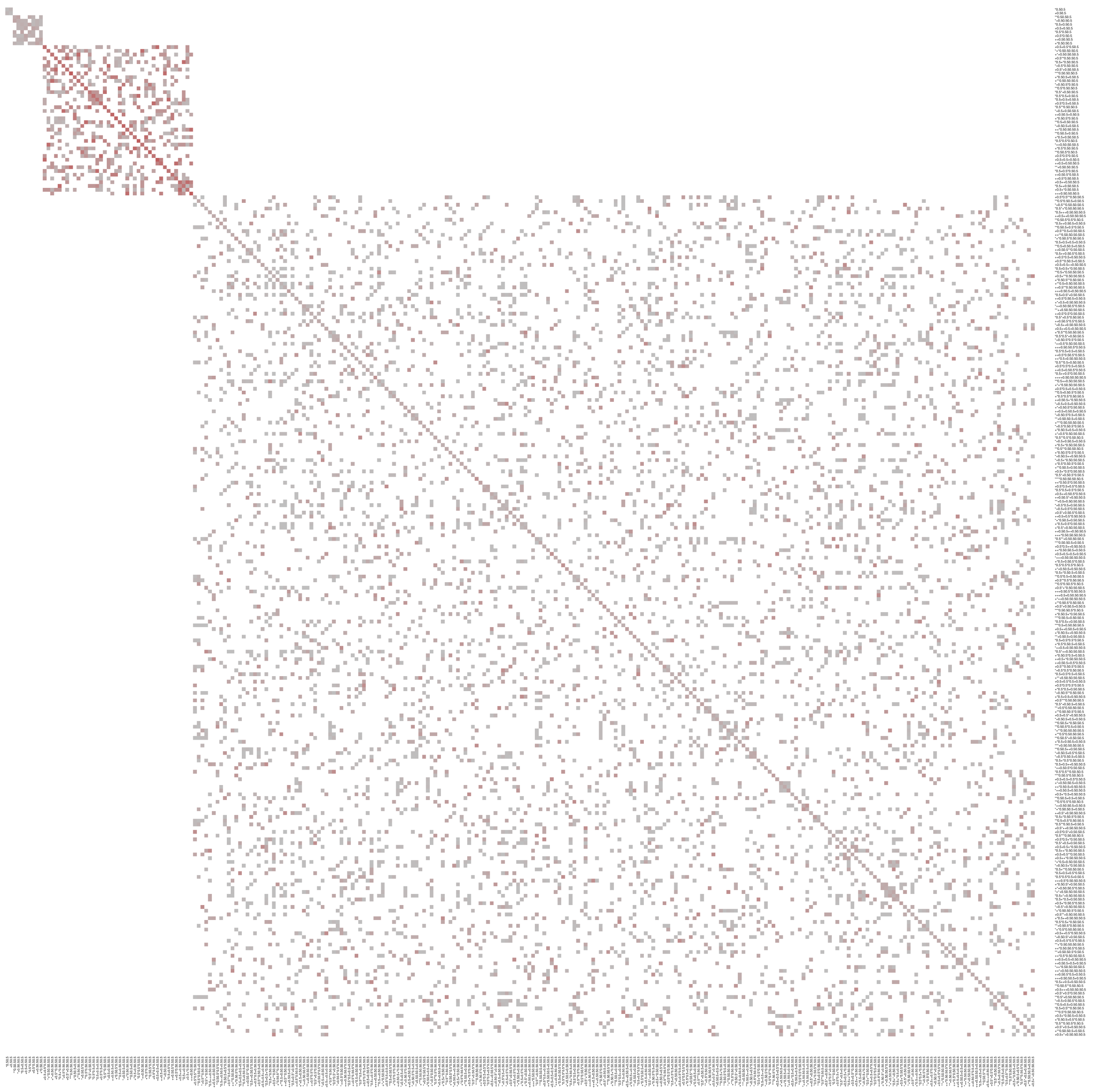


Figure 7.15: TAGE - Max frequency maps created with a maximum TAGE chromosome length of nine.

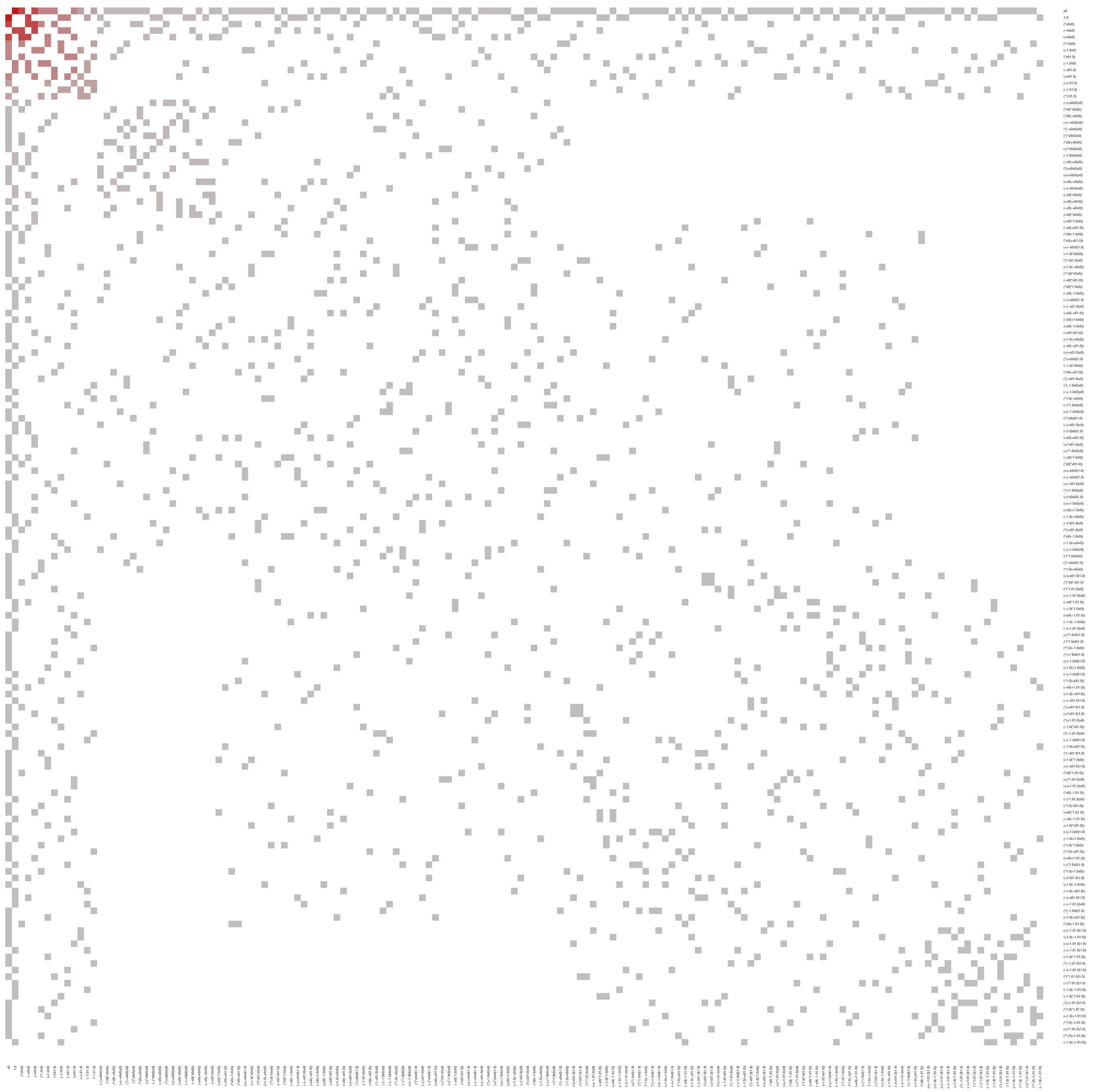


Figure 7.16: GE - Symbolic Regression frequency maps created with a maximum TAGE chromosome length of five.

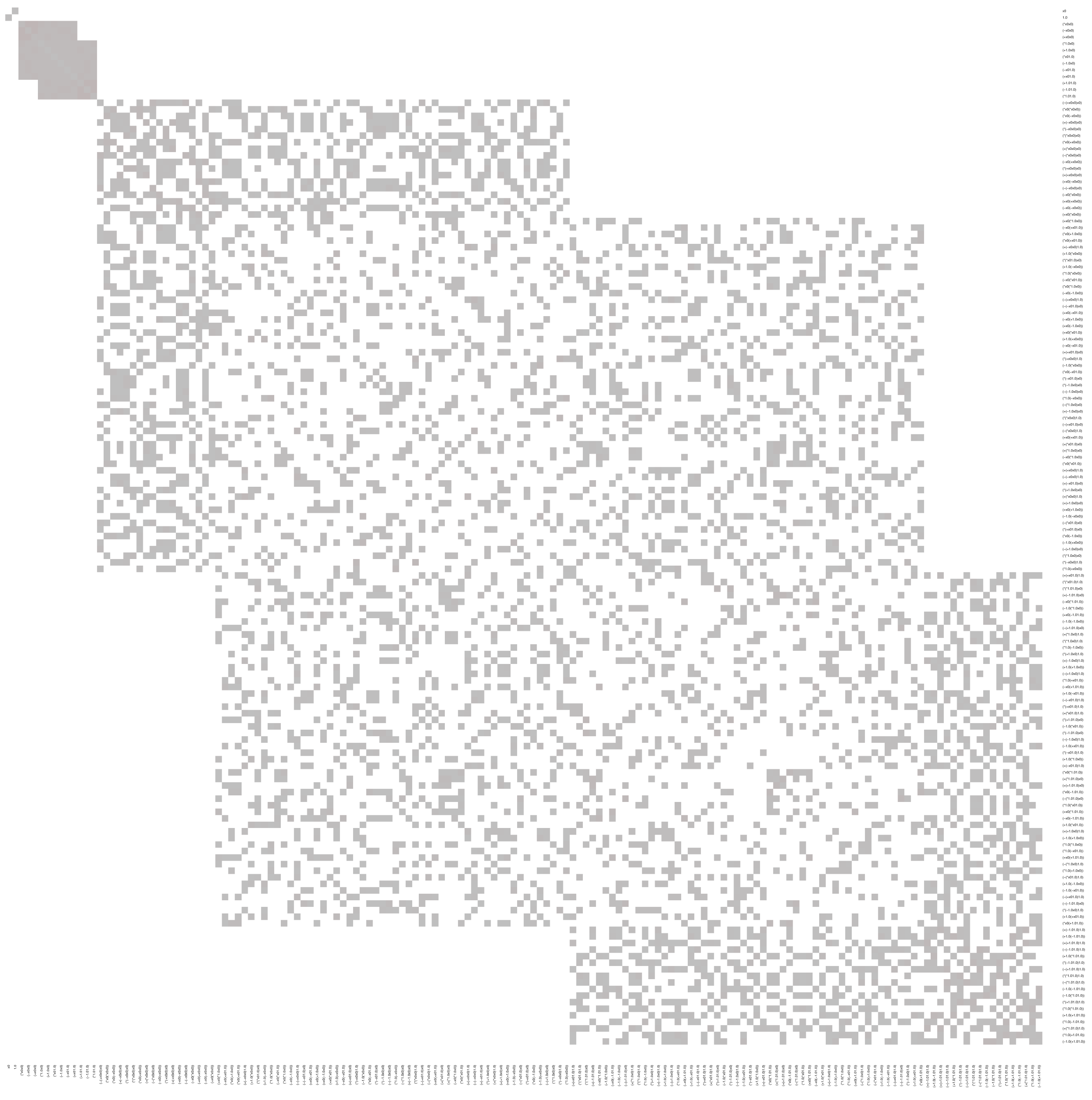


Figure 7.17: TAGE - Symbolic Regression frequency maps created with a maximum TAGE chromosome length of five.

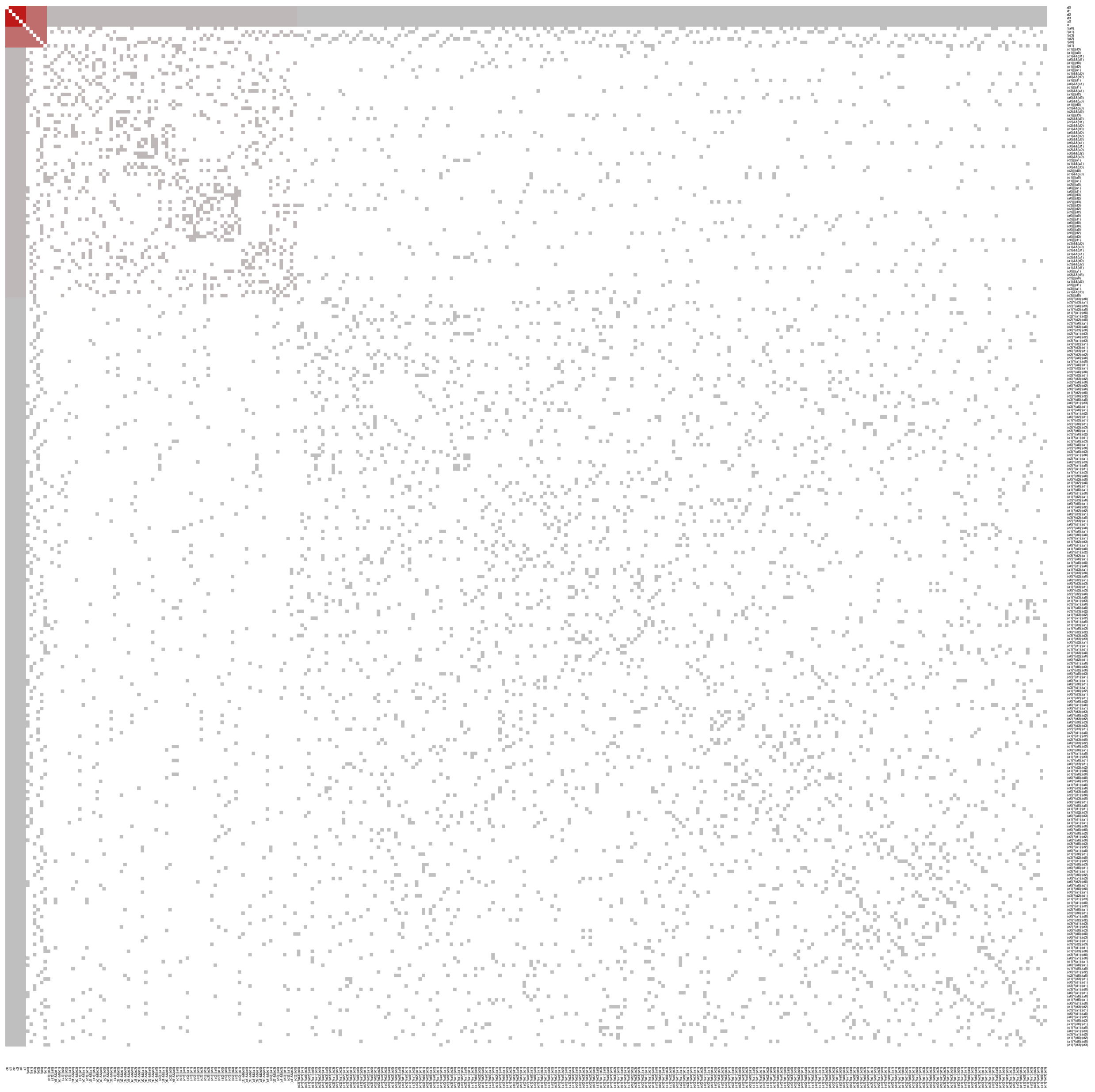


Figure 7.18: GE - Six Multiplexer frequency maps created with a maximum TAGE chromosome length of three.

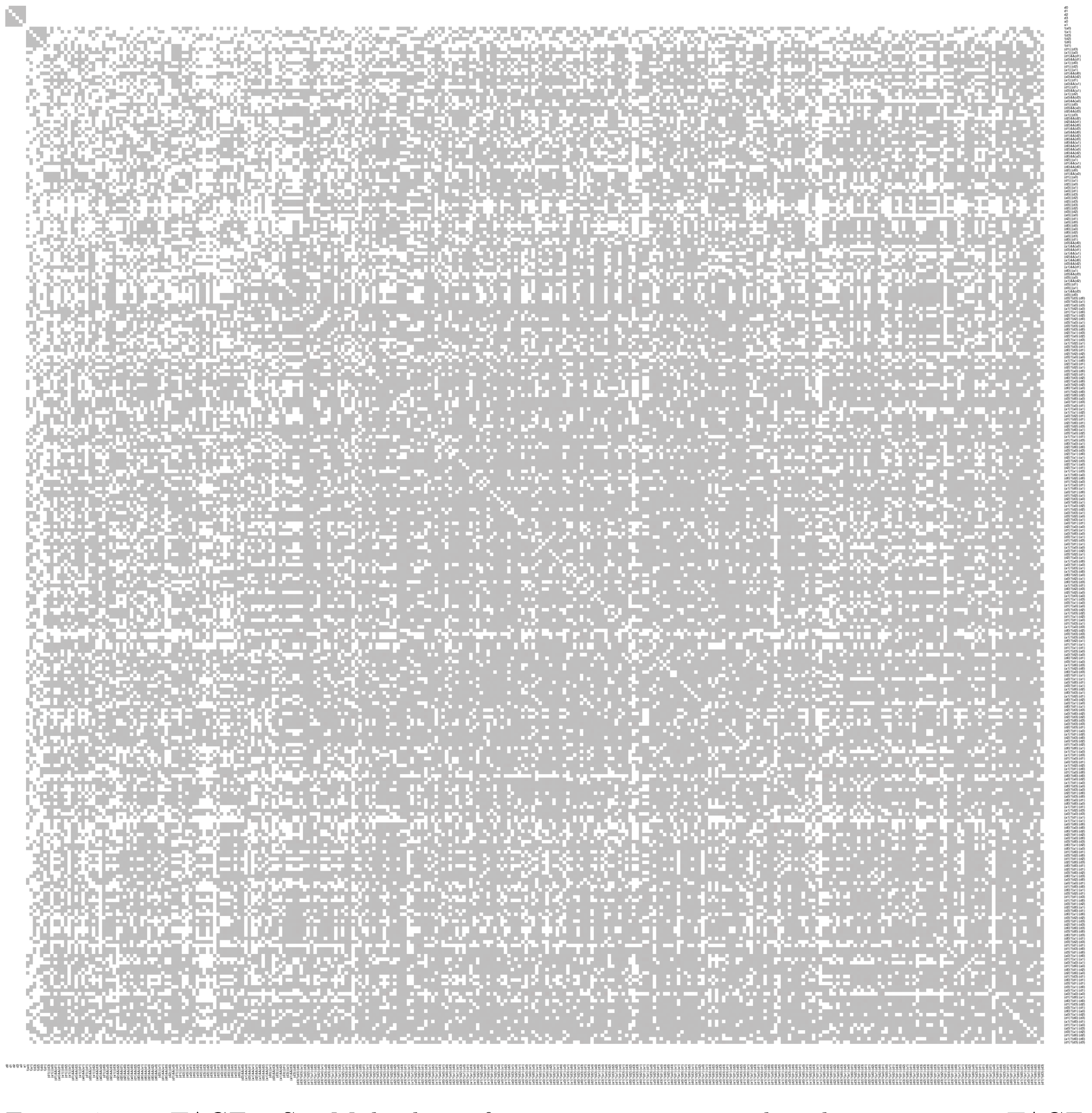


Figure 7.19: TAGE - Six Multiplexer frequency maps created with a maximum TAGE chromosome length of three.

about local optima. The high degree of connectivity visible in the TAGE CMs could be indicative of this property of increased diversity within the populations as individuals have an increased ability to move about the search space, reach more varied phenotypes than GE.

Interestingly, from the CMs it is apparent that mutation alone is not sufficient for TAGE to explore the entire search space. Unlike GE where chromosome size merely bounds the derivation tree, and hence, phenotype size, an IFM can reduce the effective size of the chromosome. This is not the case with TAGE, which makes use of the entire chromosome, and as a result IFM cannot change the size of a TAGE derivation tree or phenotype. The clusters of connections in the top left corner of each of the TAGE sub-figures are the connections between the shorter chromosomes generated during setup, the remainder of the cells are white due to the lack of connections with the phenotypes of the larger chromosome. In order to enable a full exploration of the search space, additional operators capable of changing the length of the chromosome would be needed. However, operators such as crossover act on R^2, increasing both the size and complexity of the landscapes greatly [95]. This increase in landscape complexity, a property which already causes issues in this chapter, would make visualisation very difficult.

Furthermore, the frequency maps in Figure 7.12 through Figure 7.19 show that, in GE the shorter phenotypes, produced using fewer codons, have a disproportionately high frequency of connections amongst themselves (see the red cells in the top left corner of each of the GE sub-figures), and to a lesser extent with the rest of the phenotypes (left and top borders of the GE sub-figures). In some cases the frequency of connections of these cells are orders of magnitude greater than the frequency of connections of the larger phenotypes. This indicates that the CFGs used in this study have a bias towards shorter phenotypes. A bias that does not appear in the frequency maps of TAGE's landscapes. This feature of TAGE may help avoid some of the initialisation problems experienced by GE outlined by Harper [63]. For example, when a grammar is determined to be *explosive*, randomly initialised individuals tend to be short, which has a lasting effect on the performance of the algorithm.

7.5 Summary

In this chapter, IFM landscapes are generated for a number of commonly used grammars using both CFG-based GE and TAGE. Viewing an entire landscape directly is very difficult [111], as such, the landscapes are restricted in size, and a number of different plots are employed to enable indirect analysis and comparison of the landscapes.

For the grammars examined in this chapter, it is found that phenotypes in the TAGE landscapes have a much higher degree of connectivity to the rest of the phenotypes than their counterparts in the GE landscapes. Moreover, it has been discovered that the connectivity in the TAGE landscapes is much more evenly distributed between the other phenotypes in the landscape. Whereas in the GE landscape, shorter phenotypes are much more densely connected, not only between themselves, but also to a lesser extent with the rest of the landscape.

In summary, this chapter presents a method for comparing large and highly complex landscapes using specific visualisation methods. This method of comparison can not only be further applied to the field of GE, but also to broader fields such as GP and genetic algorithms. Such an extension might enable better comparisons of each of the fields for a given problem, e.g., GP versus GE.

The following chapter provides further analysis between the two representations, concentrating on biases inherent within the representations themselves and how this effects initialisation.

Chapter 8

Initialisation and Grammar Bias

It has been shown that the form of a grammar, and indeed the language biases inherent to that grammar, can have a large impact on the performance of GBGP systems such as GE [63], amongst others [215, 82]. In Chapter 6 and Chapter 7, it is shown that modification of the grammar, an integral part of the GE algorithm, causes the algorithm to behave very differently. This is due to the ease with which the search biases in the system can change by just modifying the grammar, affecting how individuals are mapped from genotypes into phenotypes.

This chapter investigates preferential language biases which are introduced when using TAGs in GE. These biases, inherent to the representation, affect the distribution of generated derivation structures and resulting strings, both at initialisation and throughout an evolutionary run by affecting the structure of the search landscapes defined by the representation and associated variation operations. This work identifies some of these biases and examines their effect on the ability of the algorithm to solve a number of benchmark problems. The biases examined include, the structural biases imposed by the adjunction operation, as well as those introduced as a result of the grammar transformation algorithm, i.e., the loss of explicit biases imposed by the CFG when transformed into a TAG by Algorithm 3.1. A novel algorithm for addressing grammar transformation biases is proposed and a comparison between GE and TAGE is performed. The effect of these biases on the performance of the algorithm is observed and compared.

The work presented in this chapter, based on a study published by Murphy et al. [143], also considers difficulties which are encountered when attempting to compare the performance of different systems, or indeed, the same system with different representations, as is the case in this thesis. In particular, the question of fair initialisation is discussed throughout.

The layout of this chapter is as follows. Section 8.1 briefly discusses difficulties that are faced when attempting to compare different GP systems, or indeed any pair of population-based complex systems. In particular, the section discusses how these differences relate to the comparison of canonical GE and TAGE. Section 8.2 examines the importance of initialisation and gives two methods of generating similar initial derivation tree populations for both GE and TAGE. Experiments are then performed to examine the difference in performance of GE and TAGE when initialised with the same distribution of phenotype structures. In creating similar initial populations specific biases are identified. Section 8.3 examines the effect of adjunction constraint biases on TAGE, with Section 8.4 outlining language biases introduced when transforming a CFG into a TAG. A method of restoring these biases in order to investigate how they might affect TAGE is also presented. The chapter concludes with a summary in Section 8.5.

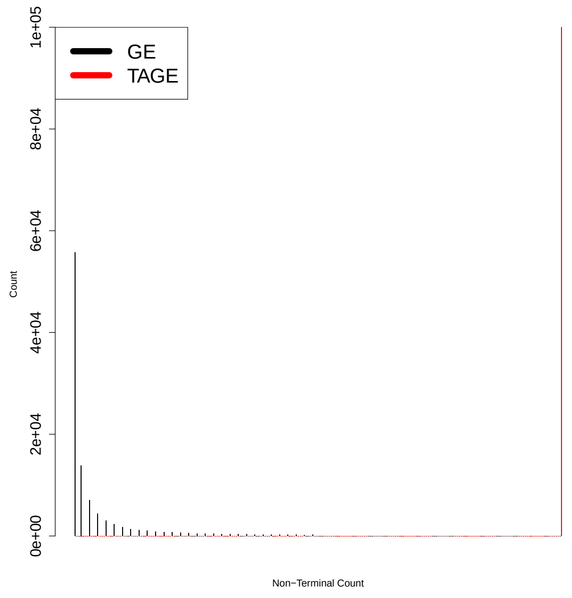
8.1 Comparing Methodologies

The problem of performing a fair comparison between different GP systems is difficult to overcome. As suggested by Hoai et al. [83], it is easy to assume that a change in behaviour observed when testing a modification to an algorithm is a direct consequence of the modification in question [38]. This is not always the case and can be a flawed assumption. GP systems are considered complex systems for this reason, and unless the modification is very localised, there can be indirect effects. This problem is especially prevalent if the modification in question results in the starting conditions of an algorithm being affected, and is further evident when comparing completely different algorithms.

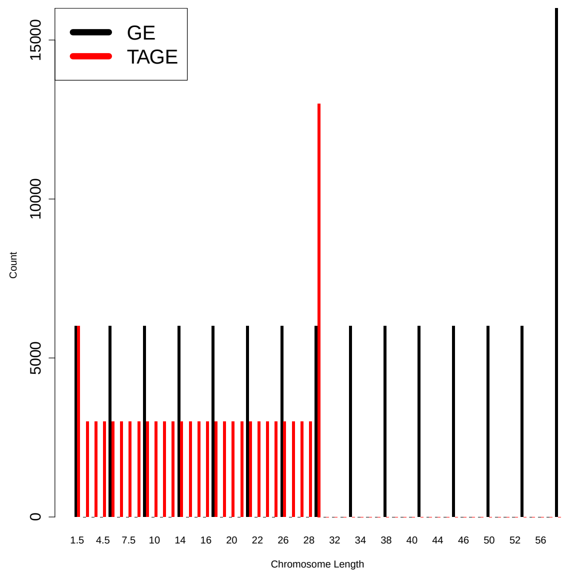
The difficulties described above can be seen when comparing standard GP algorithms with GBGP approaches, as was shown by Hoai et al. [83] when comparing GP and TAG3P. The change in representation makes it difficult to create common initial conditions for both algorithms in order to achieve a good comparison.

Comparing similar algorithms with different representations raises an interesting question: Having a common initial population, or at least initial populations drawn from similar distributions, is good practice and helps ensure a fair comparison. However, when there is a divide between genotype and phenotype, should these populations be similar in the genotypic space or in the phenotypic space? Search is performed in the genotypic space, whereas the fitness landscape lies in phenotypic space, and depending on the mapping between the two there could be a many to one relationship between genotypes and phenotypes. Creating initial populations of genotypes for two different representations, e.g., GE and TAGE, would likely result in very different populations of phenotypes. The same can be said for similar populations of phenotypes with equally different populations of genotypes. This is highlighted in Figure 8.1 for the symbolic regression CFG given in Figure 8.2c.

In the experiments presented in Chapter 6, common distributions of genotypes are used. As a counter point, this chapter investigates the use of initialisation methods which generate similar distributions of derivation trees (TAG derived trees) and phenotypes for both GE and TAGE. The following section presents these methods, examining how TAGE behaves when initialised to a distribution of phenotypes rather than genotypes, and how the comparison of performance between GE and TAGE is affected.



(a) The distributions of generated tree sizes (NT nodes) when initialising to a common genotype length.



(b) The distributions of generated genotype lengths when initialising to common distribution of derivation tree sizes.

Figure 8.1: Distributions of a) GE derivation tree/TAGE derived tree sizes and b) genotype lengths for the symbolic regression CFG given in Figure 8.2c. Due to the difference in representation, initialisation to a common distribution of one property will result in differing distributions of the other property.

```
<prog>    ::= <expr>
<expr>    ::= <expr> <expr> <op>
            | ( <expr> <expr>  <op> )
            | <var>
            | ( <var> ) <pre-op>
<pre-op> ::= !
<op>      ::= "|" | & | ^
<var>     ::= d0 | d1 | d2 | d3 | d4
```

(a) Even Five Parity

```
<prog>      ::= <code>
<code>      ::= <line> | <code> <line>
<line>      ::= <condition> | <op>
<condition> ::= if(food_ahead()==1) { <opcode> }
                else { <opcode> }
<op>        ::= left(); | right(); | move();
<opcode>    ::= <op> | <opcode> <op>
```

(b) Santa Fe Ant Trail

```
<prog> ::= <expr>
<expr> ::= ( <op> <expr> <expr> ) | <var>
<op>   ::= + | - | *
<var>  ::= x0 | 1.0
```

(c) Symbolic Regression

```
<B> ::= <B> <B> & | <B> <B> "|"
      | <B> !
      | <B> : <B> ? <B> #
      | a0 | a1 | d0| d1 | d2 | d3
```

(d) Six Multiplexer

Figure 8.2: The grammars, in Backus-Naur form, for each of the benchmark problems. Each grammar is transformed into a TAG by Algorithm 3.1 for use with TAGE. These TAGs are presented in Figure B.1 - B.4

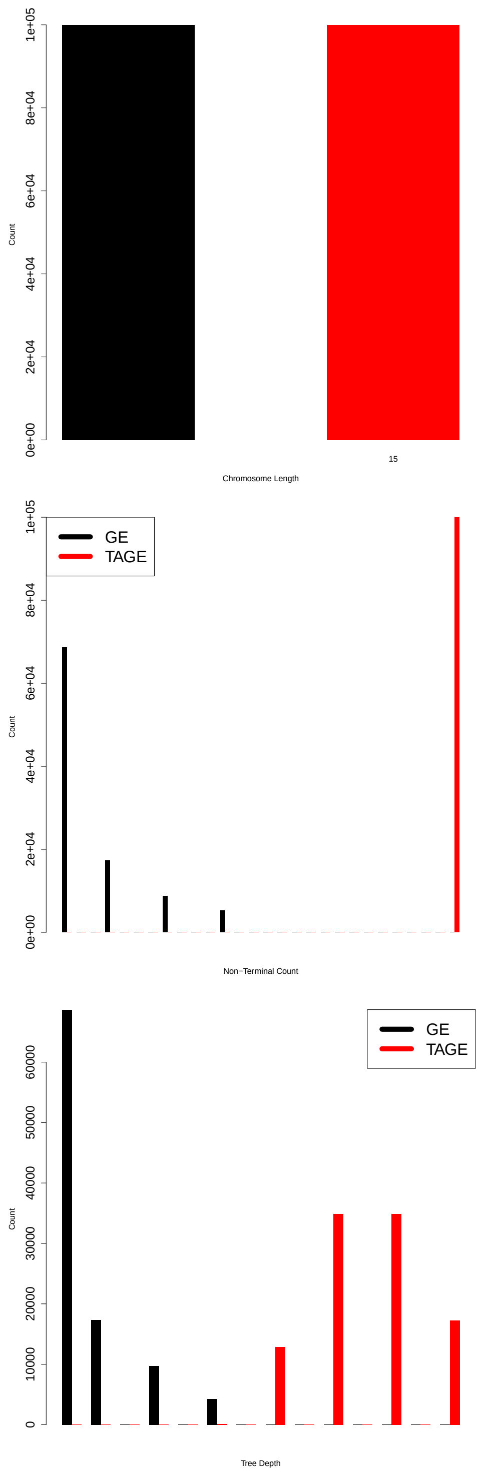
8.2 Initialisation Bias

This section examines the question of initialisation. In particular, the effect that a number of different methods of initialisation have on the behaviour of TAGE and GE. Harper [63], in examining different methods of initialisation for GE, show that the chance of achieving a successful run can be dependent upon the distribution of the initial population. Harper goes on to suggest that GE typically does not recover from a poorly distributed initial population. As such, the role of initialisation can be considered to be extremely important when evaluating the performance of GE.

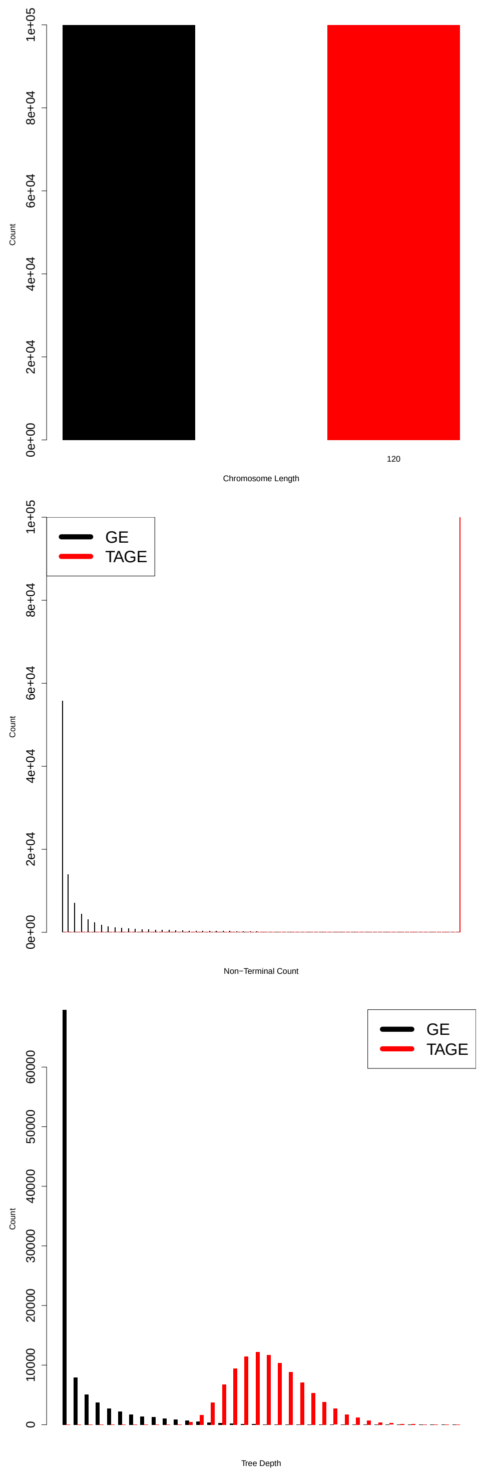
This idea is reinforced by the results presented in Section 6.2.1, particularly for TAGE, where performance decreases when the initial chromosome length is increased, and as a result, the initial phenotypes became much larger. Figure 8.3 highlights the disparity between the distributions of TAG derived trees and GE derivation trees when generated from random genotypes of the same length, as used in Section 6.2.1.

8.2.1 Generating Similar Tree Distributions

While the experiments presented in Chapter 6 make use of initial populations of similarly distributed genotypes, this section introduces two methods for generating similar distributions of derivation trees, and hence, similar distributions of initial phenotypes for both TAGE and GE. The typical method of tree initialisation in GP is the Ramped Half and Half method [113], or Sensible Initialisation in GE (see Section 2.4.1). In these approaches the population is segmented along a minimum and maximum depth interval, however, depth is not as important for GE derivation trees as it is for GP expression-trees. In GE, to ensure that there is a good distribution of phenotypes, the distribution of the number of non-terminal nodes, or tree size is more important [63]. With that in mind, a ramped tree size initialisation method is presented for GE and TAGE, with a maximum depth limit to



(a) 15 codons



(b) 120 codons

Figure 8.3: The distributions of tree sizes and depths of 100000 TAGE derived trees and GE derivation trees generated by the fixed length random genotype initialisation method used in Section 6.2.1, and the symbolic regression CFG given in Figure 8.2c.

prevent overly deep trees. This approach, which is similar to the method used by Harper [63] and PTC2 by Luke [126] without probability tables, is aimed at generating similar distributions of derivation trees for both TAGE and GE.

Ramped Non-Terminal Tree Size Initialisation (RNTS)

This method of initialisation generates derivation (and derived) trees from a uniform distribution of non-terminal node tree sizes, i.e., the number of non-terminals in each tree. Similar to sensible initialisation, a ramped approach is used, with the population segmented by the number of non-terminal nodes in the derivation trees, rather than tree depth.

For GE derivation trees, a list of non-expanded non-terminal nodes is maintained. This list is initially populated with S, the start symbol. Non-terminal nodes are uniformly removed from this list to expand. When considering a non-terminal node for expansion, the minimum number of expansions, and associated non-terminal nodes, required to terminate all remaining non-terminal nodes in the list is calculated. This value can be pre-calculated once for each non-terminal symbol in the grammar, being summed together when a total is required. This *cost* is used, in conjunction with the number of non-terminal nodes in the tree so far (*size*), to select which production rules might be applied to expand the node in question. A production rule is only considered if the minimum number of expansions/non-terminal nodes required to terminate the resulting sub-trees, summed with the current *size* and *cost*, is less than the non-terminal node count limit for the tree being generated. Recursive production rules are exclusively considered, however, if no recursive productions can be applied without requiring more than the current limit of non-terminal nodes to terminate the resulting sub-trees, then non-recursive productions are also considered. The genotype is generated retrospectively by calculating the appropriate codon values during a pre-order traversal of the generated tree. These codon values, while retaining the correct result when used for mapping, are then randomised using follow expression,

$$\begin{aligned} &r \sim U\left[0, MAX\_CODON\_VALUE - i\right] \\ &v \leftarrow r - (r \bmod c) + i \end{aligned}$$

where i is the index of the chosen production rule, c is the total number of production rules, and v is the generated codon value.

This process is emulated for TAGE derived trees. To generate an individual, an initial tree is chosen at random, and a list of available adjoinable addresses is maintained, pre-populated with the addresses from the chosen initial tree. An address of a non-terminal node in the derived tree is chosen at random and removed from this list. The maximum depth that the sub-derived trees rooted by this node reach is calculated. A list of auxiliary trees to adjoin at that node is then generated. An auxiliary tree is only considered if its root node is labelled with the same symbol as the chosen node, if the addition of the auxiliary tree does not cause the current non-terminal size limit to be broken, and if the depth of the tree's foot node summed with the maximum sub-tree depth previously calculated is less than the maximum depth limit. This process is repeated until no further adjunction operations can be performed without breaking the non-terminal node size limit. In this case, the correct randomised codon values can be calculated as each address is selected from the list and after each auxiliary tree has been chosen.

Figure 8.4 shows the distributions of a number of properties for 100000 individuals generated using this approach for GE and TAGE, with a non-terminal node count interval of $[21, 70]$ and a maximum depth threshold of 10. As this method generates similar distributions of phenotypes for the two representations, the opposite of the trend observed in Figure 8.3 can be seen, with the distribution of non-terminal node counts being similar and uniformly distributed along the provided interval, while the distributions of genotypes differ substantially.

From the distributions of generated tree depths, it can be seen that while the method successfully generates a somewhat uniform distribution for GE, the distribution of generated TAGE derived trees are heavily biased towards deeper trees. This is reinforced by the distribution of tree shapes. This metric reports the percentage of non-terminal nodes in the tree used to represent the left side of the tree. Here, the distribution of tree shapes for TAGE trees appears bi-modal in nature, skewed towards extremes of the metric.

While the generated trees for both GE and TAGE have resulted in similar uniform distributions of non-terminal node counts, their respective distributions of tree depth and shape differ. Even though the grammars used in generating these trees are equivalent, there are biases inherent to TAGs which affect the probabilities of certain structures being generated, as well as biases inherent to CFGs that are not preserved by this grammar transformation. Two such biases in the TAGE representation have been identified. Specifically, these are an adjunction bias, biases imposed upon the language by the choice of adjunction nodes, and a grammar transformation bias, introduced when transforming from one grammar type to another. These biases are examined in Section 8.3 and Section 8.4 respectively. Next, a method of tree generation for producing identical populations for use with both representations is described, mitigating the differences between them, at least during initialisation.

Ramped Random TAGE Chromosome Initialisation (RRT)

As is shown in the previous section, it is difficult to guarantee equal distributions of initial populations when using different representations. However, if the two representations do not differ completely from each other, sharing some intermediate state of representation, then it may be possible to generate a single population which can be used by both representations. Hao et al. [61] achieve this by generating an initial population using TAG3P+,

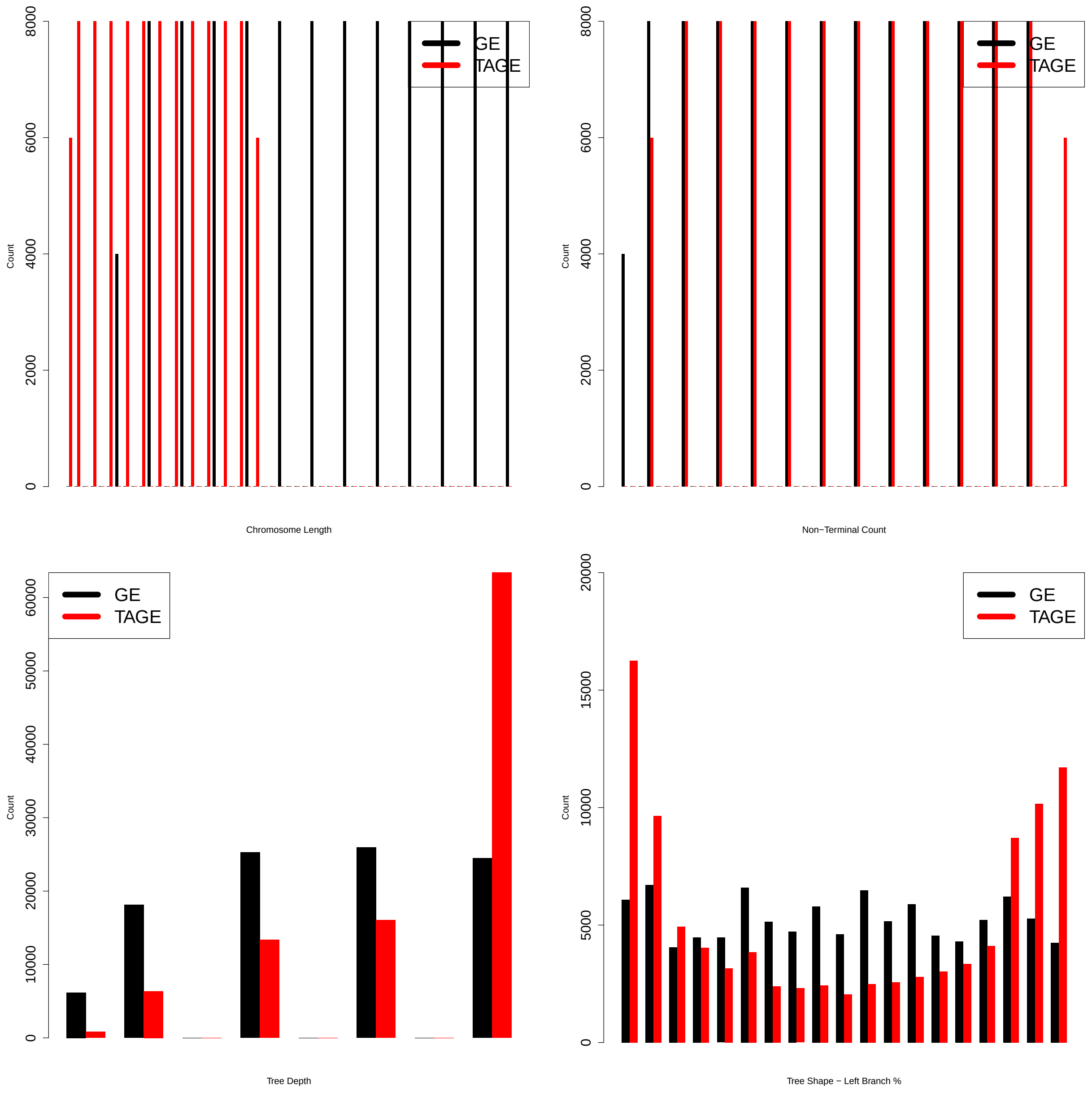


Figure 8.4: Histograms of properties of 100000 individuals generated from similar distributions of non-terminal node counts (RNTS) for both TAGE and GE using the symbolic regression CFG given in Figure 8.2c. The resulting differing distributions of genotypes can be seen along with differing distributions of tree depths and sizes.

with the resulting derivation trees being used directly by CFG-GP [215].

A similar approach is possible with TAGE for GE. While reverse-mapping TAGE phenotypes to GE chromosomes is very difficult, a more feasible approach is to take the initialised TAGE derivation trees and to generate their respective derived trees. As the size of a TAGE derivation tree is directly proportional to the length of its genome, it is trivial to generate a uniform distribution of tree sizes. Moreover, the GE chromosome can be easily calculated retrospectively from TAGE derived trees (see Algorithm C.1). It is more difficult to calculate a TAGE chromosome from a GE derivation/TAGE derived tree, as there can be a many to one mapping from TAGE derived tree to TAGE derivation tree.

As the size of a TAGE individual is directly proportional to the length of its chromosome, a simple ramped approach is taken. The population is segmented over an interval of chromosome lengths, and a random chromosome of a length from that interval is generated for each individual.

Figure 8.5 shows the distributions of a number of properties for 100000 individuals generated by this approach for GE and TAGE. A chromosome length interval of $[1, 30]$ is used. A similar trend to the previous approach is seen in Figure 8.5. The distribution of non-terminal node counts are similar, as well as being uniformly distributed along the provided interval, while the distributions of genotypes differ substantially. The distributions of all of the tree-based metrics match for both GE and TAGE populations as the generated trees for each representation are identical, it is only the genotypes that encode these trees that differ. While a uniform distribution of tree shapes is desirable, this approach also suffers from the bi-modal distribution of tree shapes seen previously in the TAGE populations in Figure 8.4. Such distributions can be undesirable as they can indicate a bias towards the generation of a particular tree shape, whereas the solution may be of a shape that is less likely to be generated. Examination of biases that have been identified as being responsible for this distribution of tree shapes is presented later in Section 8.3 and Section 8.4.

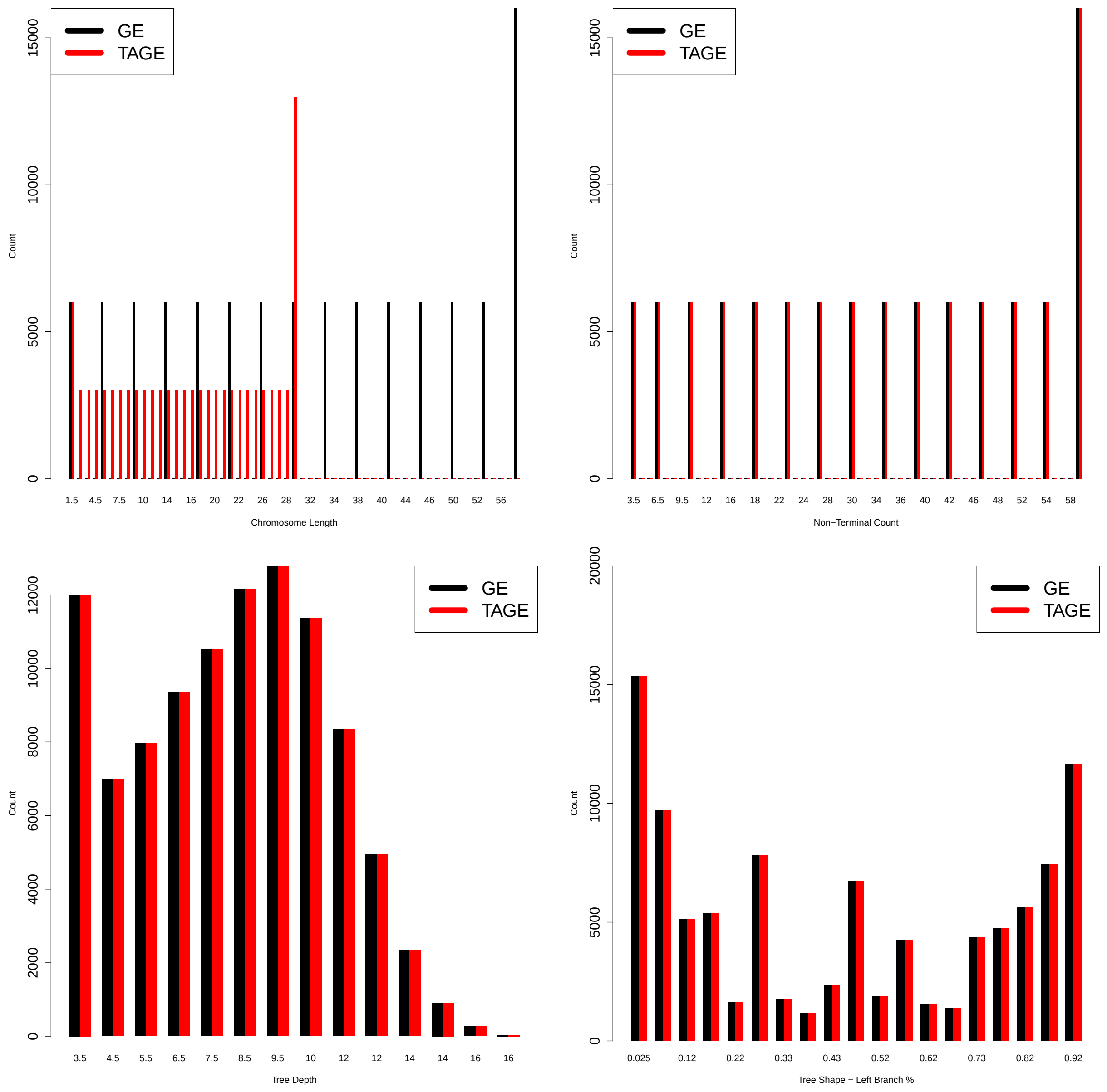


Figure 8.5: Histograms of properties of 100000 individuals produced from randomly generated TAGE chromosomes with the TAG created from the symbolic regression CFG given in Figure 8.2c. GE chromosomes to produce the trees can be generated retrospectively by careful traversal of the trees (see Algorithm C.1).

8.2.2 Experimental Setup

The focus of this section is to investigate the behaviour of TAGE and GE when initialised with populations generated from different methods and distributions. The experiments are concerned with three such methods, each of which distributes the initial populations differently throughout the search space. These methods are: random genotypes of a fixed length (RX, where X is the chromosome length), as are used throughout Chapter 6; a ramped non-terminal node tree size approach (RNTS) which generates similar distributions of trees for both representations (Section 8.2.1); and a ramped random TAGE genotype approach (RRT) which generates identical populations for both GE and TAGE (Section 8.2.1).

The experiments first consider the question of whether the method of initialisation has a significant impact on the performance of either GE or TAGE. The performance, μ, of any one setup is taken as the mean of the distribution of its best fitness values after the final generation. A setup, a, performs better than another setup, b, if $\mu_a < \mu_b$, and if there is a significant difference in their distributions of best fitness values. $<$ is used, as fitness is minimised throughout this thesis.

The performance of the different setups considered are, μ_{G0} and μ_{G1}, for R015 and R120 respectively, μ_{G2}, for RNTS, and μ_{G3}, for RRT. μ_{T0}, μ_{T1}, μ_{T2} and μ_{T3} are the same for TAGE respectively. With this in mind, the following null hypothesis, H_0, and accompanying alternate hypothesis, H_1, are stated:

H_0 : None of the initialisation methods have a significant impact on the performance of GE or TAGE, i.e., $\mu_{G0} = \mu_{G1} = \mu_{G2} = \mu_{G3}$ and $\mu_{T0} = \mu_{T1} = \mu_{T2} = \mu_{T3}$.

H_1 : At least one of the initialisation methods has a significant impact on the performance of GE or TAGE, i.e., $\neg\left(\mu_{G0} = \mu_{G1} = \mu_{G2} = \mu_{G3}\right)$ or $\neg\left(\mu_{T0} = \mu_{T1} = \mu_{T2} = \mu_{T3}\right)$.

α : 0.05 using the Mann-Whitney U, two-tailed, non-parametric test.

Subsequent to these experiments, an additional set of hypotheses are also tested to ascertain the effects of initialising both representations with the same populations:

H_{a0} : There is no difference in performance between TAGE and GE when TAGE is initialised with the same population of phenotypes as standard GE, i.e., $\mu_{G3} = \mu_{T3}$.

H_{a1} : There is a difference in performance between TAGE and GE when TAGE is initialised with the same population of phenotypes as standard GE, i.e., $\mu_{G3} < \mu_{T3}$ or $\mu_{T3} < \mu_{G3}$.

α : 0.05 using the Mann-Whitney U, two-tailed, non-parametric test.

Experimental Parameters

Five hundred independent runs are performed with GE and TAGE using the GEVA software [180] on each of the four benchmark problems outlined in Section 6.1.3. The grammars used are reproduced in Figure 8.2 for convenience. These CFGs are transformed into TAGs for use with TAGE by means of Algorithm 3.1. The generated TAGs are provided in Figure B.1 - B.4. The evolutionary parameters adopted for all of the experiments are presented in Table 8.1. Effective region crossover and population validation, as discussed in Section 6.2.3 and Section 6.2.2 respectively, are utilised with standard GE for all experiments performed in this section.

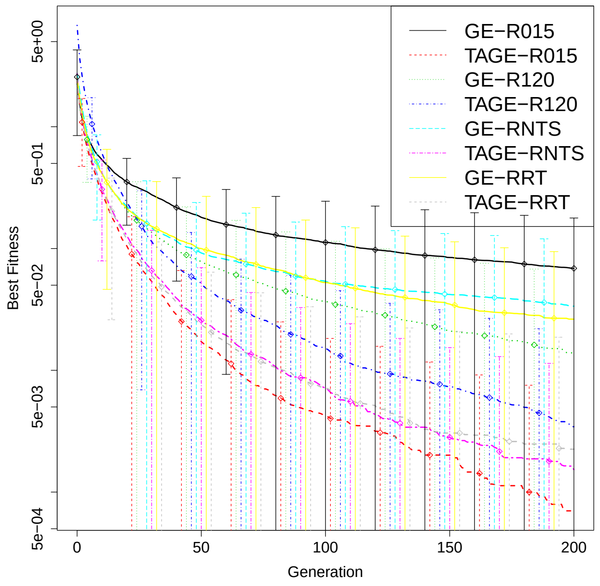
8.2.3 Results

Initial Hypotheses

Figure 8.6 presents the mean best and mean average population fitness plots for each of the initialisation methods on the symbolic regression problem. Plots for all problems are provided in Figure C.1 and Figure C.2. The resulting μ values for each experiment are reported in Table 8.3. All GE setups are compared to each other, independently

Table 8.1: GE parameters.

Parameter	Value
System	GEVA v1.1 (extended)
Random Number Generator	Mersenne Twister MT199973
Generations	200
Population Size	100
Initialisation Method 0	Random (R015,R120)
Initial Chromosome Size	15, 120
Initialisation Method 1	RampedNTSize (RNTS)
Initial NT Size Interval	$[21, 70]$
Initial Max Tree Depth	10
Initialisation Method 2	RampedRandomTAG (RRT)
Initial Chromosome Size Interval	$[1, 30]$
Crossover Point Selection	Effective Region Only
Population Validation	Replace with Parent
Max Chromosome Wraps	0
Replacement Strategy	Generational
Elitism	10 Individuals
Selection Operation	Tournament
Tournament Size	3
One Point Crossover Probability	0.9
Integer Mutation Probability	0.02



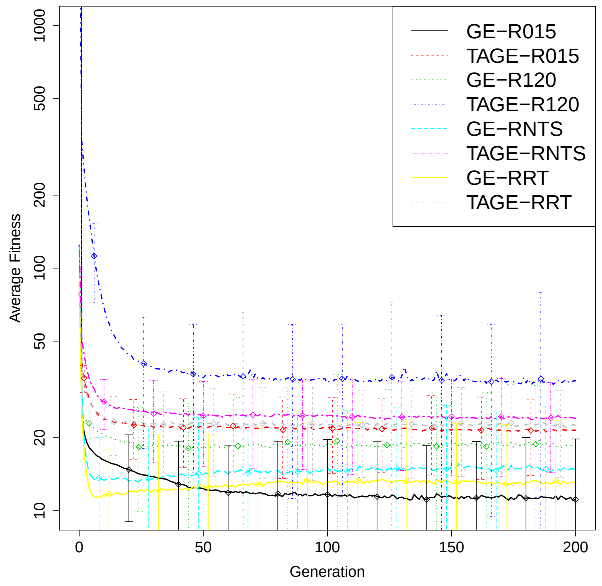


Figure 8.6: Mean best (left) and mean average population (right) fitness plots for the symbolic regression problem with error bars of one standard deviation. Plots for the remaining problems can be see in Figure C.1 and Figure C.2.

or all TAGE setups being compared to each other. A Kruskal-Wallis test is performed on each group to determine if there is a significant difference within the group, i.e., the populations making up the group are non-identical. The results of these test are given in Table 8.2. As before, the Mann-Whitney U, two-tailed, non-parametric test is used, however, the resulting p-values are corrected for the group comparison using Bonferroni correction. Symmetric comparisons, and self comparisons are ignored for this correction. Shaded cells indicating which comparisons between GE setups, and separately, between TAGE setups, are significant. It is clear from Table 8.3 that H_0 is rejected for both TAGE and GE, as many different comparisons of initialisation methods result in significantly different distributions of best fitness values at the final generation.

Table 8.2: Results (p-values) of the Kruskal-Wallis test for each group independently compared. Highlighted cells indicate that the grouped populations are non-identical.

Problem	GE	TAGE
Even	8.2486e-03	1.6183e-05
SF	1.6759e-18	9.8777e-31
SR	3.4692e-05	2.7677e-16
6 Mux	1.7591e-02	2.7454e-17

Secondary Hypotheses

The results of the comparison between TAGE and GE, using the same initial populations (RRT), are reported here. The resulting μ values for each experiment are reported in Table 8.4, with shaded cells indicating which comparisons between GE and TAGE are significant.

Even Five Parity Mean best fitness and average fitness plots can be seen in Figure 8.7. The mean best fitness values, μ, for GE and TAGE are 0.394 and 0.058 respectively. The Mann-Whitney U p-value is 1.7091e-07, below the threshold value of 0.05, thus rejecting the null hypothesis H_{a0} for this setup. The performance values are listed in Table 8.4, and all p-values are presented in Table C.2.

Ant Trail Mean best fitness and average fitness plots can be seen in Figure 8.8. The mean best fitness values for GE and TAGE are 9.964 and 9.708 respectively. The Mann-Whitney U p-value of 0.1535 is above the threshold value of 0.05, and hence, the null hypothesis H_{a0} cannot be rejected for this problem. The performance values are listed in Table 8.4, and all p-values are presented in Table C.2.

Symbolic Regression Mean best fitness and average fitness plots can be seen in Figure 8.9. The mean best fitness values for GE and TAGE are 0.0262812 and 0.002251577 respectively. The Mann-Whitney U p-value is 1.3339e-12, below the threshold value

Table 8.3: Separate comparisons of the performance of GE and TAGE using different methods of initialisation. RN015 and RN120 are fixed random initialisation of length 15 and 120. RNTS is ramped non-terminal size initialisation and RRT is ramped random TAGE chromosome initialisation. GE and TAGE are compared separately. The mean best fitness values, μ, and respective standard deviation values, are reported for each setup. GE methods are compared with other GE methods. TAGE methods are compared with other TAGE methods. Mann-Whitney U test results comparing distributions of best fitness values at generation 200 are also presented. Shaded cells indicate p-values < 0.05 for that comparison. The labels have been omitted from the horizontal axis of the grids. They are, in order of left to right: R015. R120, RNTS and RRT. All p-values are provided in Table C.1.

GE							TAGE						
μ (SD)	Successes					Method	μ (SD)	Successes					
0.1940 (1.0151)	480	╲	▨	▨	▨	R015	0.0320 (0.3567)	496	╲	▨		
0.3620 (1.2674)	454	▨	╲			R120	0.1440 (0.7101)	475	▨	╲	▨	▨
0.3200 (1.1542)	459	▨		╲		RNTS	0.0420 (0.3585)	492		▨	╲	
0.3940 (1.4024)	457	▨			╲	RRT	0.0580 (0.5507)	493		▨		╲

(a) Even five parity

GE							TAGE						
μ (SD)	Successes					Method	μ (SD)	Successes					
16.004 (12.773)	99	╲	▨	▨	▨	R015	9.130 (10.658)	224	╲	▨	▨	
10.012 (11.539)	178	▨	╲	▨		R120	17.776 (13.779)	113	▨	╲	▨	▨
13.200 (11.776)	134	▨	▨	╲	▨	RNTS	11.544 (11.594)	182	▨	▨	╲	
9.964 (11.629)	175	▨		▨	╲	RRT	9.708 (10.860)	211		▨		╲

(b) Santa Fe ant trail

GE							TAGE						
μ (SD)	Successes					Method	μ (SD)	Successes					
0.0688 (0.1094)	312	╲	▨		▨	R015	0.0007 (0.0057)	489	╲	▨		
0.0138 (0.0470)	414	▨	╲	▨		R120	0.0034 (0.0154)	467	▨	╲	▨	▨
0.0336 (0.0820)	350		▨	╲		RNTS	0.0015 (0.0087)	481		▨	╲	
0.0263 (0.0673)	362	▨			╲	RRT	0.0023 (0.0162)	485		▨		╲

(c) Symbolic regression

GE							TAGE						
μ (SD)	Successes					Method	μ (SD)	Successes					
4.6580 (3.9674)	154	╲			▨	R015	0.7100 (1.9103)	423	╲	▨		
4.5140 (3.6814)	133		╲			R120	1.3120 (2.2461)	337	▨	╲	▨	▨
4.4900 (3.3747)	115			╲	▨	RNTS	0.6220 (1.6113)	417		▨	╲	
3.9500 (3.7228)	185	▨		▨	╲	RRT	0.5140 (1.6356)	442		▨		╲

(d) Six mulitplexer

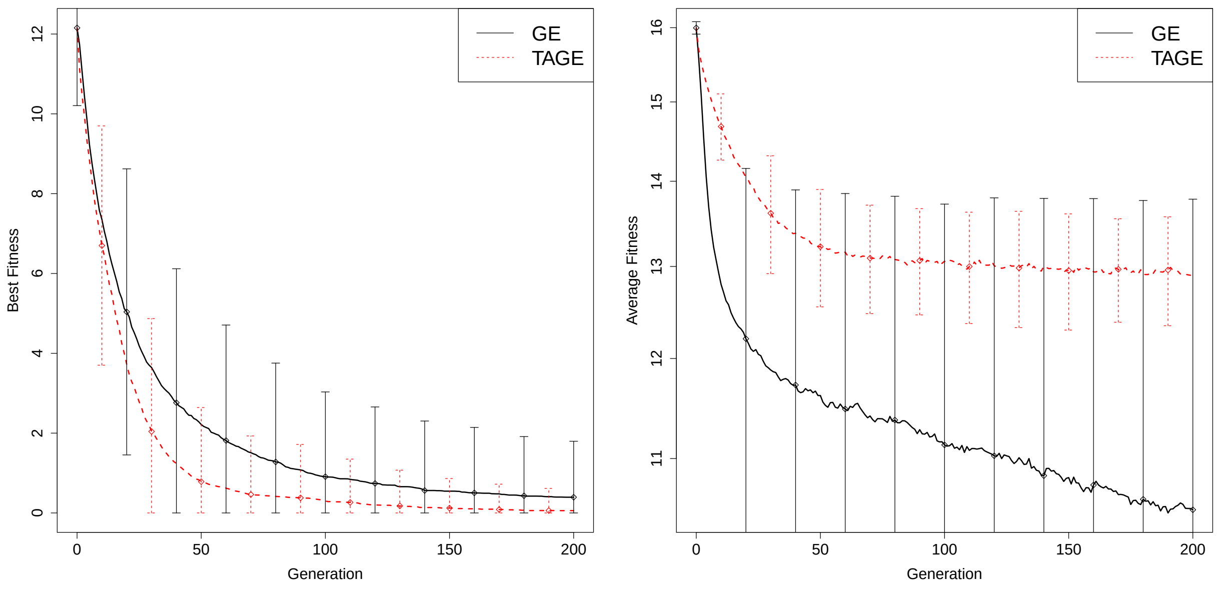


Figure 8.7: Mean best (left) and mean average population (right) fitness plots for the even five parity problem with error bars of one standard deviation.

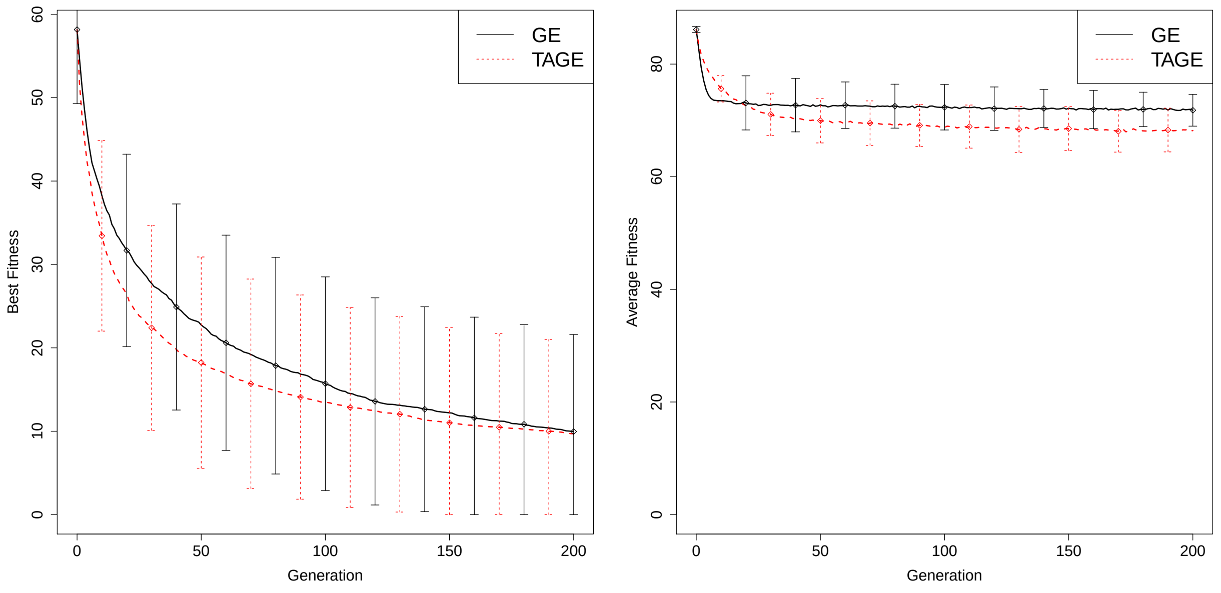


Figure 8.8: Mean best (left) and mean average population (right) fitness plots for the Santa Fe ant trail problem with error bars of one standard deviation.

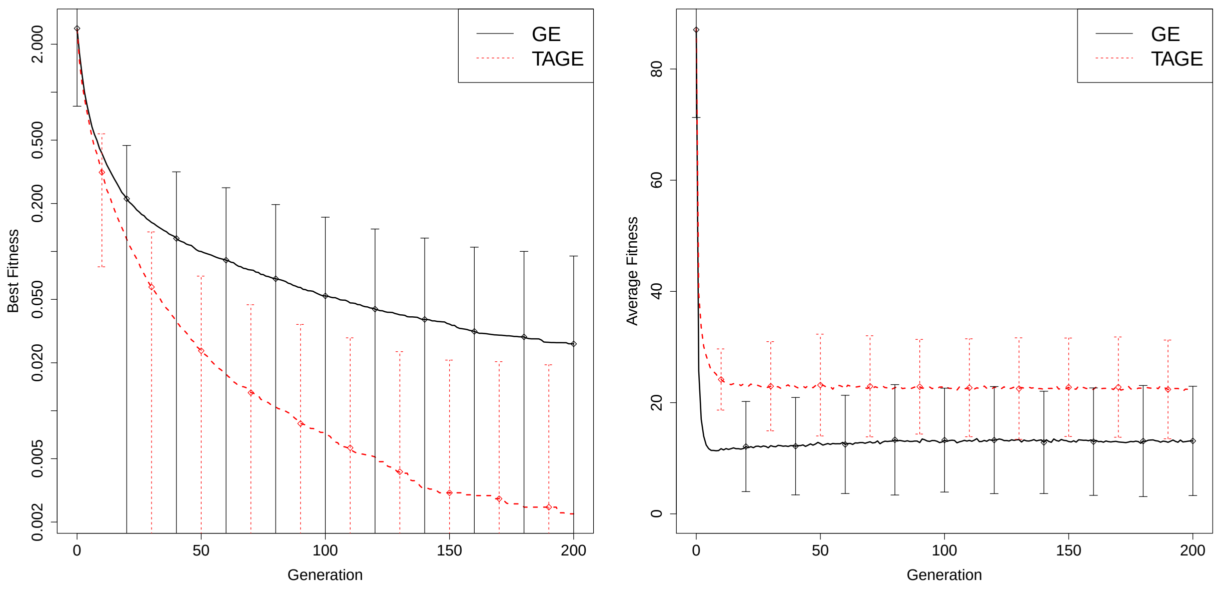


Figure 8.9: Mean best (left) and mean average population (right) fitness plots for the quartic symbolic regression problem with error bars of one standard deviation. A log scale is used for the y-axis of the mean best fitness plot.

of 0.05, therefore rejecting the null hypothesis H_{a0} for this setup. The performance values are listed in Table 8.4, and all p-values are presented in Table C.2.

Six Multiplexer Mean best fitness and average fitness plots can be seen in Figure 8.10. The mean best fitness values for GE and TAGE are 3.95 and 0.514 respectively. The Mann-Whitney U p-value is 1.4397e-65, below the threshold value of 0.05, rejecting the null hypothesis H_{a0} for this setup. The performance values are listed in Table 8.4, and all p-values are presented in Table C.2.

8.2.4 Discussion

While it is clear from Table 8.3 that the distribution of the initial population throughout the search space can result in significantly different outcomes in terms of performance, it is not clear, by comparing the means, μ, if any one method provides an advantage

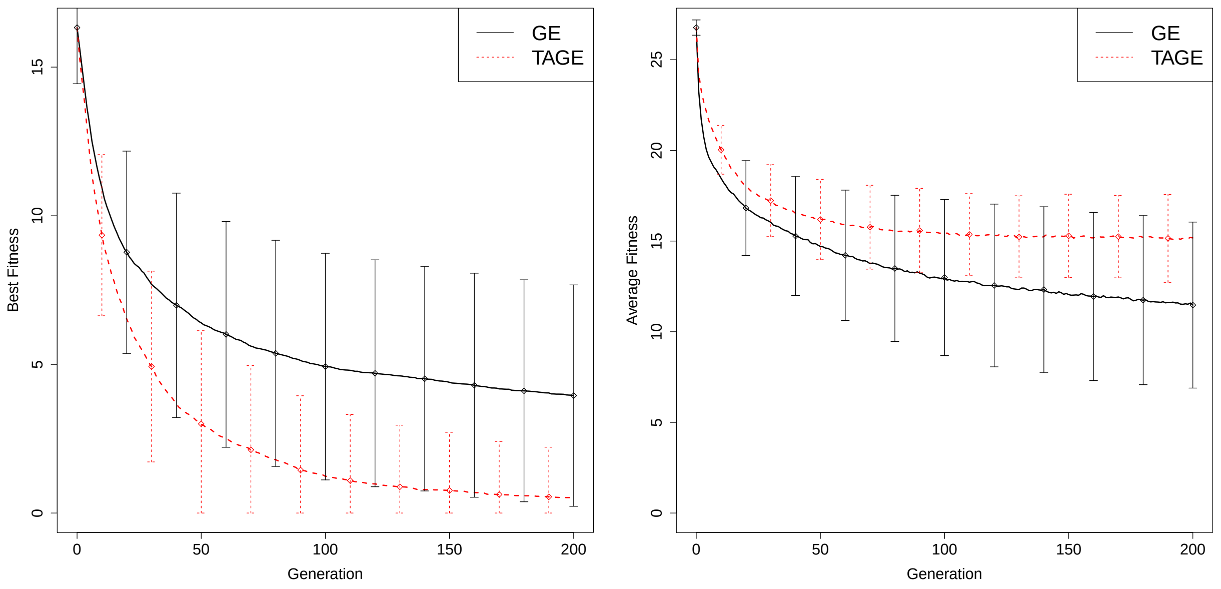


Figure 8.10: Mean best (left) and mean average population (right) fitness plots for the six multiplexer problem with error bars of one standard deviation.

Table 8.4: A comparison of results obtained by GE and TAGE across the benchmark problems using the same initial population with a broad distribution of derivation tree sizes generated by the method described in Section 8.2.1. The mean best fitness values across 500 runs for the final generation and the total number of successful runs are reported, correct to four decimal places. Mann-Whitney U test results comparing distributions of best fitness values at generation 200 are also presented. Shaded cells indicate p-values $<$ 0.05. p-values are provided in Table C.2.

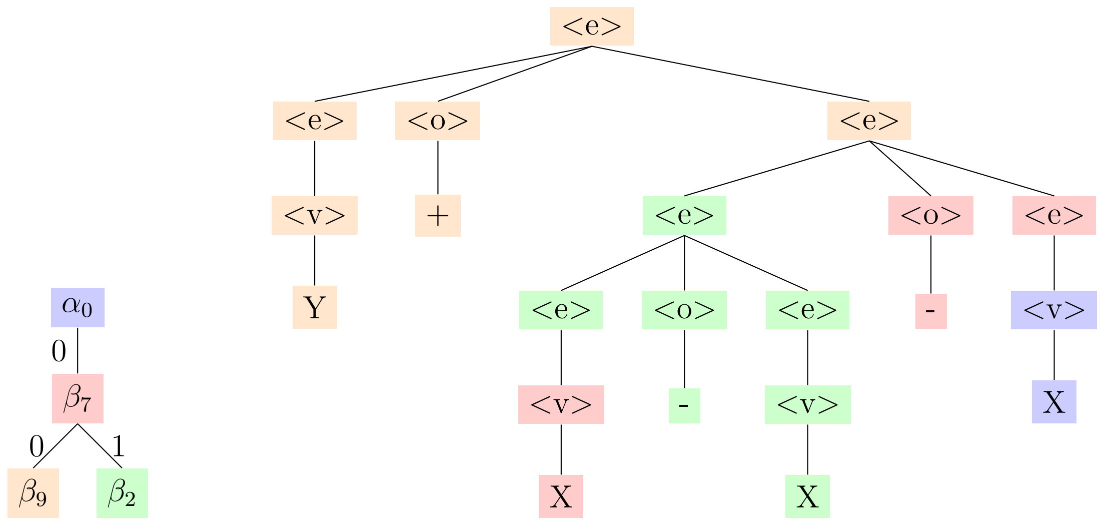
	GE		TAGE	
	μ_{G3} (SD)	Successes	μ_{T3} (SD)	Successes
EFP	0.3940 (1.4024)	457	0.0580 (0.5507)	493
SF	9.9640 (11.6290)	175	9.7080 (10.8598)	211
SR	0.0263 (0.0673)	362	0.0023 (0.0162)	485
6 Mux	3.9500 (3.7228)	185	0.5140 (1.6356)	442

over another. The application of an approach to one problem might result in significantly different outcomes over other approaches, as well as better μ values and an increased number of successful solutions, e.g., RRT for GE on the six multiplexer problem. However, the application of the same approach on a different problem can also result in little or no improvement over other the other methods, e.g., RTT for GE on the even five parity problem.

In Chapter 6, the experiments performed reveal that GE and TAGE perform significantly different when initialised using a common distribution of initial genotypes, the second set of experiments presented here investigate if that difference in performance does not exist when using identical initial distributions of derivation trees. Table 8.4 shows that this is false for three of the four problems, rejecting the null hypothesis, H_{a0}, that there is no difference in performance between TAGE and GE when initialised with the same population of phenotypes as standard GE for those problems. The TAGE experiments result in better μ values than GE for all problems, in addition to finding more successful solutions. Interestingly, similar trends are seen in the mean average fitness plots for all of the experiments which reject H_{a0}. TAGE mean population fitness values are much farther away from the optimum than those of GE. This trend is also observed in the experiments presented in Chapter 6. The results presented here allow the difference between GE and TAGE in the dispersion of the initial populations within the search space to be disqualified as a contributing factor to this lack of convergence towards the optimum apparent in TAGE.

8.3 Adjunction Bias

While TAGs have been shown to be both weakly and strongly equivalent to CFGs [96], the composition operations used with TAGs can impose preferential biases on the probability



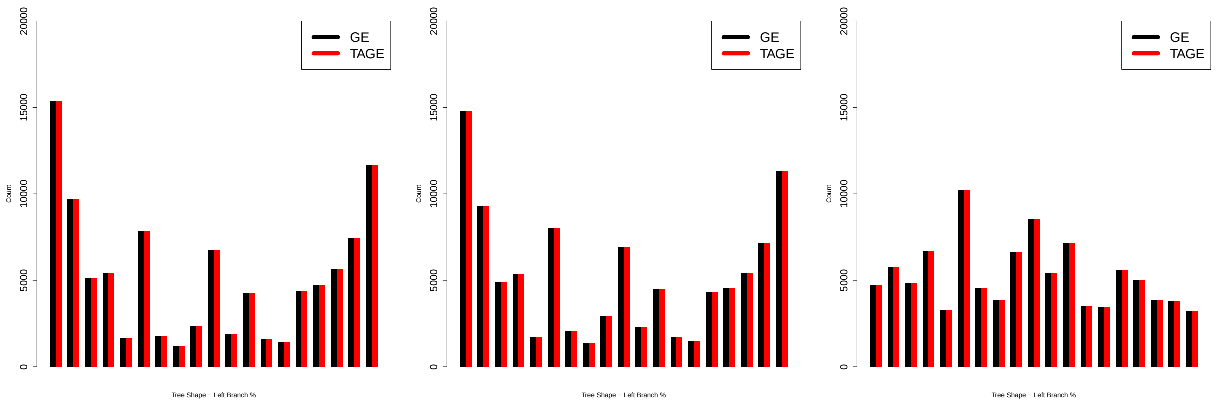
(a) TAG Derivation Tree. (b) TAG Derived Tree.

Figure 8.11: An example of a TAG derivation tree and its respective derived tree. Adjunction restrictions can result in the promotion of unbalanced tree shapes.

of particular structures being generated. Various different types of biases have been applied to TAGs using different adjunction constraints [99]. Hoai et al. [82] make use of a form of restricted adjunction in order to apply biases to the generation of particular structures in the TAG3P+ system in an attempt to improve performance.

In Section 5.2, adjoinable nodes are defined as any node which has not been adjoined to, and is labelled with a symbol which is also the label of an auxiliary tree root node. The section goes on to say that adjunction is restricted from occurring on foot nodes. While this restriction makes the representation easier to comprehend and to implement, such a restriction imposes a bias on the mapping process, i.e., once an adjunction is performed, the sub-tree attached to the foot node of the new auxiliary tree can no longer be modified. This reduces the chance of certain shapes being generated, e.g., from the existing derivation tree given in Figure 8.11, it is not possible to adjoin an additional auxiliary tree into node address 3 of β_7, the right-most `<e>`, when restricting foot node adjunction. Such a derived tree could only be constructed using a different sequence of adjunctions.

This restriction appears to contribute only partially towards the bi-modal distributions



(a) Original adjunction (b) Foot node enabled (c) Root node restricted

Figure 8.12: The effect of various adjunction restrictions on the distributions of generated derived tree shapes. Beginning from the original bias, foot node adjunction is enabled, and following this root adjunction is then restricted.

of tree shapes that are presented in Section 8.2. Figure 8.12b highlights the slight effect that removing the restriction and the resulting bias from the representation has on the distribution of generated tree shapes.

While imposed biases can be a hindrance, they can also be beneficial [82, 215]. With regards to the restrictions on the adjunction operation, by restricting adjunction at the root nodes of auxiliary trees, a further, much more pronounced, improvement is seen in the distribution of tree shapes produced. Without this restriction, if at any point during derivation an adjunction is performed on the top-most adjoinable node, usually the root of an auxiliary tree, then the sub-derived tree, rooted at this node, becomes the sub-tree of the new auxiliary tree's foot node, with the remainder of this new auxiliary tree off to one side. The effect of this is that the shape of the tree can become heavily skewed to one side, and as the adjunction sites are uniformly chosen, the likelihood of that partial auxiliary tree being expanded is small, $\frac{n}{N-n}$, where n is the number of adjoinable nodes in the auxiliary tree branch and N is the number of adjoinable nodes in the entire the tree. By eliminating adjunction at the root nodes of auxiliary trees, a more uniform distribution of generated tree shapes can be achieved. This can be seen in Figure 8.12c which highlights the effect that removing the restriction and the resulting bias from the representation has on the distribution of tree shapes.

8.3.1 Experimental Setup

In order to ascertain the effect of adjunction restrictions on the performance of TAGE, the experimental setup and the definition of performance from Section 8.2.2 are reused here with a single initialisation method, the ramped random TAGE chromosome approach (RRT) which generates identical initial populations for both GE and TAGE. Adjunction restrictions are preferential biases in the grammar. These biases not only affect the generation of the initial populations, but also the structure of the search space, affecting the algorithm throughout the run. To investigate the effect of this bias the following null hypothesis, H_{b0}, and accompanying alternate hypothesis, H_{b1}, are examined:

H_{b0} : Removing the foot node adjunction constraint and adding an adjunction constraint to root nodes has no effect on the performance of TAGE.

H_{b1} : Removing the foot node adjunction constraint and adding an adjunction constraint to root nodes has an effect on the performance of TAGE.

α : 0.05 using the Mann-Whitney U, two-tailed, non-parametric test.

Additional experiments are performed to re-assess the comparison between GE and TAGE with this change of adjunction biases in mind. To this end, the following null hypothesis, H_{c0}, and accompanying alternate hypothesis, H_{c1}, are stated:

H_{c0} : There is no difference in performance between GE and TAGE when using the modified adjunction constraints and common initial populations.

H_{c1} : There is a difference in performance between GE and TAGE when using the modified adjunction constraints and common initial populations.

α : 0.05 using the Mann-Whitney U, two-tailed, non-parametric test.

8.3.2 Results

The results of the first set of experiments, examining H_{b0} by comparing TAGE with and without the modified adjunction restrictions, are presented in Table 8.5. From this table it can be seen that H_{b0} cannot be rejected for three of the four problems examined. Only for the Santa Fe ant trail problem is the comparison of μ significant, rejecting H_{b0} for that problem. The p-values for these comparisons are given in Table C.3. The average derived tree size and tree depth plots for each problem are given in Figure 8.13 and Figure 8.14, showing the effect of the modified adjunction constraints on the trees over the course of the runs.

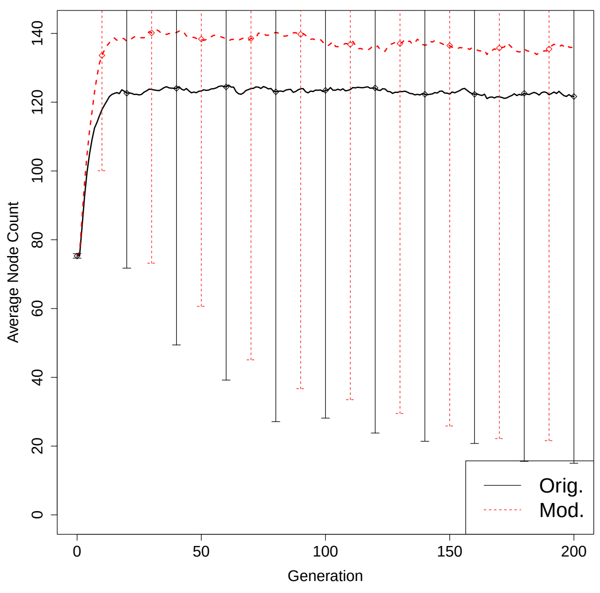
The results of the second set of experiments, examining H_{c0} are presented in Table 8.6. H_{c0} is rejected for all problems examined. TAGE achieves better μ values for three of the four problems. GE achieves a better value for μ in the Santa Fe problem. The results for *Orig.*, TAGE with unmodified adjunction constraints, are included in this table for convenience.

Table 8.5: A comparison of results obtained by TAGE when modifying adjunction biases across the benchmark problems. *Orig.* indicates the original adjunction restrictions presented in Table 8.4 and reproduced here for comparison. *Mod.* indicates the modified adjunction restrictions. The mean best fitness values across 500 runs for the final generation, μ and the total number of successful runs are reported, correct to four decimal places. Mann-Whitney U test results comparing distributions of best fitness values for *Orig.* and *Mod.* at generation 200 are also presented. Shaded cells indicate p-values < 0.05. p-values for the comparisons are provided in Table C.3.

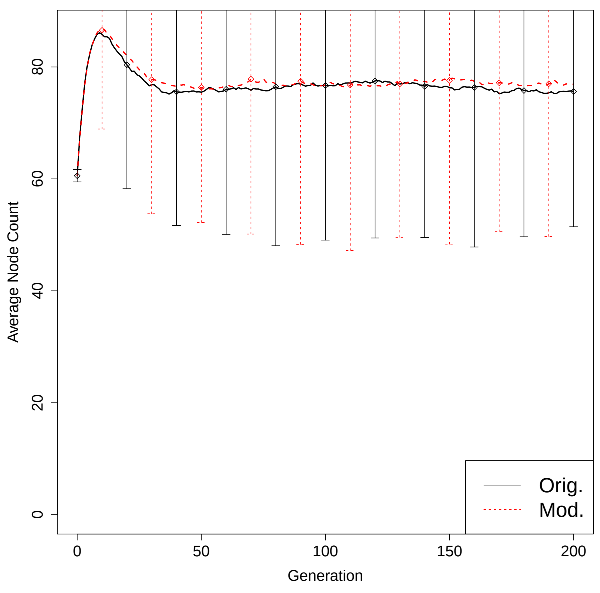
	Orig.		Mod.	
	μ (SD)	Successes	μ (SD)	Successes
EFP	0.0580 (0.5507)	493	0.0700 (0.4956)	489
SF	9.7080 (10.8598)	211	11.3200 (11.4714)	187
SR	0.0023 (0.0162)	485	0.0015 (0.0103)	482
6 Mux	0.5140 (1.6356)	442	0.3800 (1.1638)	435

Table 8.6: A comparison of results obtained by GE and TAGE when modifying adjunction biases across the benchmark problems using the same initial population with a broad distribution of derivation tree sizes generated by RRT (Section 8.2.1). *Orig.* indicates the original adjunction restrictions presented in Table 8.4 and reproduced here for comparison. *Mod.* indicates the modified adjunction restrictions. The mean best fitness values across 500 runs for the final generation, μ and the total number of successful runs are reported, correct to four decimal places. Mann-Whitney U test results comparing distributions of best fitness values for *Mod.* at generation 200 are also presented. Shaded cells indicate p-values < 0.05. p-values for *Mod.* comparisons are provided in Table C.4, while *Orig.* are provided in Table C.2.

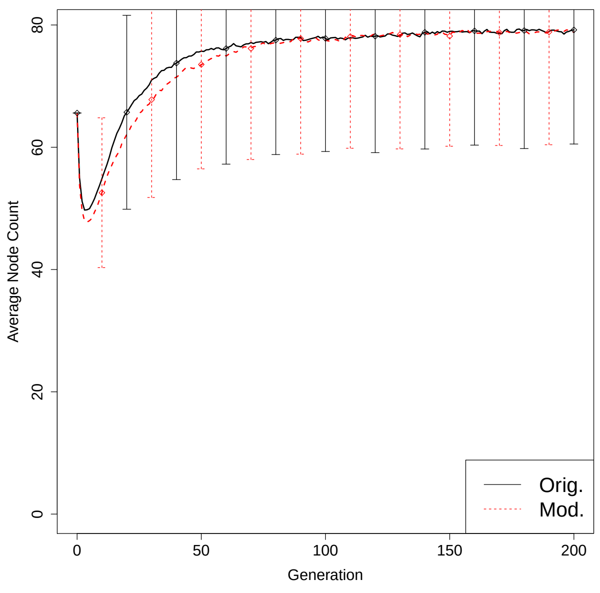
		GE		TAGE	
		μ_{G3} (SD)	Successes	μ_{T3} (SD)	Successes
EFP	Orig.	0.3940 (1.4024)	457	0.0580 (0.5507)	493
	Mod.	0.2920 (1.1685)	466	0.0700 (0.4956)	489
SF	Orig.	9.9640 (11.6290)	175	9.7080 (10.8598)	211
	Mod.	8.8000 (11.2177)	211	11.3200 (11.4714)	187
SR	Orig.	0.0263 (0.0673)	362	0.0023 (0.0162)	485
	Mod.	0.0331 (0.0781)	349	0.0015 (0.0103)	482
6 Mux	Orig.	3.9500 (3.7228)	185	0.5140 (1.6356)	442
	Mod.	2.708 (3.2011)	249	0.3800 (1.1638)	435



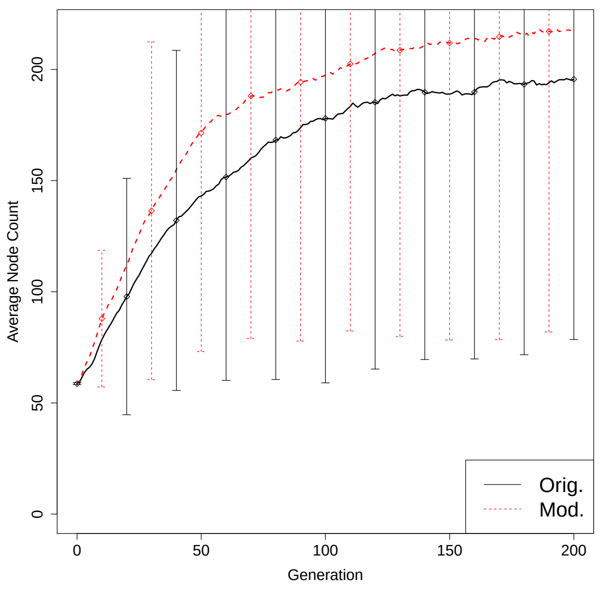
(a) Even five parity



(b) Santa Fe ant trail

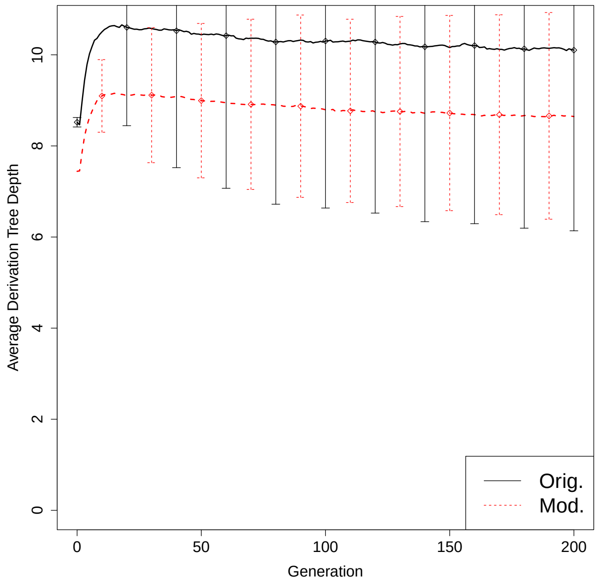


(c) Symbolic regression

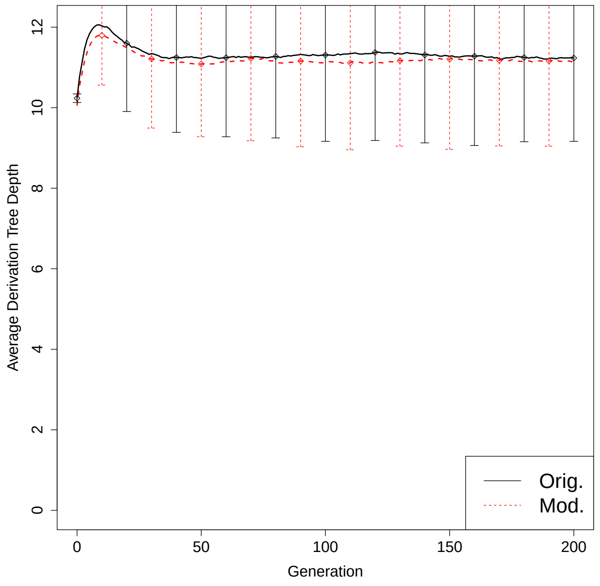


(d) Six multiplexer

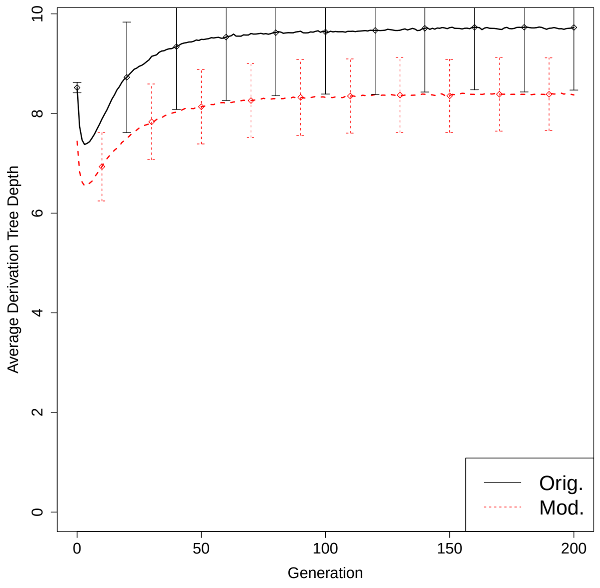
Figure 8.13: Average TAGE derived tree size plots comparing the original adjunction restrictions with the modified restrictions on all problems with error bars of one standard deviation.



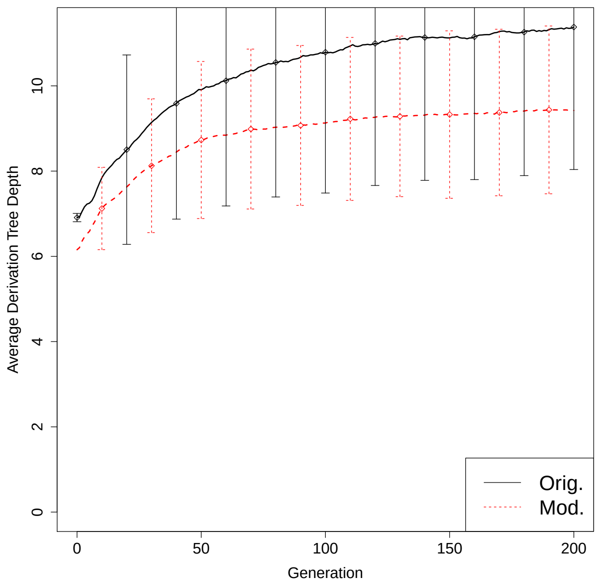
(a) Even five parity



(b) Santa Fe ant trail



(c) Symbolic regression



(d) Six multiplexer

Figure 8.14: Average TAGE derived tree depth plots comparing the original adjunction restrictions with the modified restrictions on all problems with error bars of one standard deviation.

8.3.3 Discussion

While the modification of adjunction biases improves the distribution of tree shapes, as is shown in Figure 8.12c, the modification does not see a significant difference in the performance of TAGE in three of the four problems, see Table 8.5. For the Santa Fe ant trail problem, a significant difference in the distribution of best fitness values is shown, with the mean best fitness value, μ, moving further from the optimum with the modification of adjunction biases.

Figure 8.13 and Figure 8.14 show that the three problems for which the adjunction bias modification did not significantly affect performance, are affected in terms of their distributions of tree sizes and depths. With the average tree sizes increasing, or staying the same, and the average depths decreasing, indicating a distribution of trees centred about a mean of trees that are more full. The difference in depths, seen in Figure 8.14, between the original adjunction restrictions and the modified restrictions strengthens the hypothesis that the distribution of unmodified tree shapes shown in Figure 8.12a, which are bi-modal in nature, are largely caused by single adjunctions at the root nodes, with a good distribution of tree shapes attached to the foot nodes of these “root” auxiliary trees. This difference, for the even five parity and symbolic regression problems, in particular, appears to be the same as the extra depth an adjunction at the root of a derived tree adds to the total tree depth.

Interestingly, while the Santa Fe problem is the only problem to be significantly affected by the bias modifications discussed above, Santa Fe is also the only problem not be affected in terms of average generated tree sizes and tree depths over the course of the experiments, as shown in Figure 8.13 and Figure 8.14. This is as a result of the grammar used for Santa Fe, in particular the recursive rules.

```
<code>       ::= <line> | <code> <line>
```

```
<opcode>     ::= <op> | <opcode> <op>
```

The auxiliary trees generated from these rules, unlike the other grammars, have a single adjoinable address each.

```
<expr>    ::= <expr> <expr> <op>
```

The result of the modifications presented in this section move this address from the root node to the foot node without affecting the resulting tree shape. While the average tree sizes and tree depths are not affected, μ has decreased and the distribution of mean best fitness values has significantly changed, indicating that the original biases provide some benefit to TAGE when solving this problem.

Separately, in examining the comparison between GE and TAGE using the same initial populations, populations whose distribution of tree shapes and depths have been affected by the modifications, Table 8.6 shows that there is now a significant difference in the performance between GE and TAGE at the final generation on all four of the problems. TAGE has better mean values, μ, for even five parity, symbolic regression and six multiplexer. While the initial populations are affected for both GE and TAGE, that is where the effects of these modifications stop for GE. With that in mind, GE manages to find many more successful solutions with the modified initial population than before for the symbolic regression and six multiplexer problems.

These examinations highlight the fact that GE is highly complex and while some biases may seem disadvantageous, they can be beneficial depending on the representation and the problem being addressed.

8.4 Grammar Transformation Bias

This section examines the loss of preferential biases when transforming from CFG to TAG. The probability of a specific terminal production being selected when generating a sentence

```
<code>  := <value>
<value> := <digit> | x
<digit> := 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

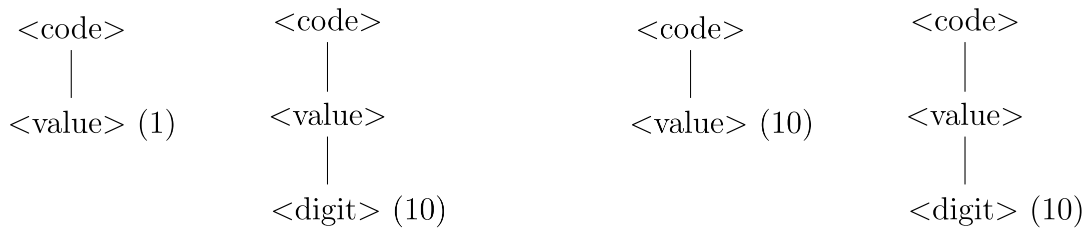
Figure 8.15: A sample CFG for illustrating the loss of bias

from a CFG depends, not only upon the probability of that terminal production being selected within its own rule, but also upon the probability of each preceding expansion occurring within the derivation sequence. When transforming a CFG into a TAG, these biases are lost, and while this can be seen as a feature of the TAG representation, it can have unexpected effects for certain types of grammar and should be kept in mind if designing a CFG that may be transformed into a TAG.

For example, when generating a sentence from the CFG presented in Figure 8.15 whose derivation does not contain any recursive expansions, when expanding `<value>` there is a 0.5 chance of selecting `x` or `<digit>`. If `<digit>` is selected, there is then a 0.1 chance of selecting any one of the digits, zero to nine. From this, it can be seen that even though there are 11 different words which might be generated, there is an equal probability of ending up with either an `x` or any one of the ten digits each. Once the CFG is transformed into a TAG, there are 11 initial trees to chose from, one with an `x` on the frontier and ten with a digit. Consequently, there is a $0.\overline{09}$ chance of generating any specific word, as opposed to a 0.5 chance for the `x` or a 0.05 chance for one of the digits when using the CFG. This can make it difficult for certain words to be generated, both during initialisation and throughout a run.

8.4.1 Probabilistic Tree-Adjoining Grammatical Evolution

It is possible to account for these lost biases, and as such, a novel method is developed which examines the probabilities contained within a CFG and applies them to the transformed TAG. This is method is known as Probabilistic Tree-Adjoining Grammatical Evo-



(a) TAGE stubs (b) PTAGE stubs

Figure 8.16: TAGE tree stubs for the CFG in Figure 8.15 a) with a $0.\overline{09}$ chance of selecting x, and b) equivalent PTAGE stubs with a 0.5 chance.

lution (PTAGE). The concept, which is similar in theory to the application of structural and lexical biases to improve search by Hoai et al. [82] with TAG3P+, is to recreate, in the transformed TAGs, the preferential biases for the different structures and resulting sequences of terminal symbols present in the CFGs used generate those TAGs.

As described in Section 5.4, the sets of initial and auxiliary trees are not generated in full at the beginning of the TAGE algorithm, but rather sets of elementary tree *stubs* are generated. A tree stub is an almost fully expanded elementary tree, with the terminal symbol leaf nodes excluded. In place of each terminal symbol is a number representing the total amount of different terminal nodes, *variations*, that might be attached at that point to complete the tree. For example, continuing with the sample grammar from Figure 8.15 above, rather than the eleven initial trees described above, there would be two stubs. The first with the number `1` rather than an `x`, and a second with the number `10` rather than ten different trees, each with one of the digits as a leaf node. The resulting tree stubs are shown in Figure 8.16a.

PTAGE examines the CFG and calculates the probability of specific production rules being chosen during expansion. This is shown in Algorithm 8.1. Once the initial stubs have been created, PTAGE then looks at the stubs which share production rule choices, and uses the ratio of the probabilities of selecting those productions over others within each rule to calculate scaling factors for the total variations for each stub. This process is shown in detail in Algorithm 8.2. Finally, the variations at each non-terminal leaf node in the stub are updated to reflect this new total, see Algorithm 8.3. In this example, since there should be an equal chance of generating an `x` as a digit, the variations on that stub are updated from `1` to `10`, as shown in Figure 8.16b. The effect of this is that when selecting a tree there are now 20 trees to chose from, ten `x` trees and ten digit trees, one for each number.

Algorithm 8.1 Calculating production selection probabilities for a CFG.

```
Require: $G$ {A CFG}
  {Create a map productions to probabilities for each non-terminal symbol}
  $probs$ ←new Map<Symbol, Map<Production, Double>>
  for all $r_i$ in $G$.getRules do
    $map \leftarrow probs$.get($r_i$.getLHS)
    for all $p_j \in r_i$ do
      if $p_j$.isTerminal then
        $existingProb \leftarrow map$.get($\epsilon$)
        $map$.put($\epsilon$, $existingProb + 1.0/r_i$.size)
      else
        $existingProb \leftarrow map$.get($p$)
        $map$.put($p$, $existingProb + 1.0/r_i$.size)
  return $probs$
```

8.4.2 Experimental Setup

To explore the effect of restoring the CFG biases on the generated TAGs used by TAGE, the experimental setup from Section 8.2.2 is repeated here using a single initialisation method, RRT, for both TAGE and PTAGE. The two setups are compared on four benchmark problems, with the grammars given in Figure 8.2, and the following null hypothesis, H_{d0},

Algorithm 8.2 Calculate the total variations for each TAGE tree stub to match CFG probabilities.

```
Require: stubs {List of a group of similar tree stubs with the same indexed symbol}
Require: nodeIndex {All stubs are similar from this node up to the root}
Require: variations {Map of elementary tree to respective variation counts}
Require: probs {CFG probabilities from Algorithm 8.1}
  uniqueProductionsToStubs ← new map<Production, List<ElementaryTree>>
  uniqueProductionsToVariationCounts ← new map<Production, Int>
  {Group all stubs who expand from this node using the same production, and calculate total variations for each unique
  production}
  for all $tree_i$ ∈ stubs do
      node ← $tree_i$.getNode(nodeIndex)
      if node = $tree_i$.getFootNode then
          continue
      prod ← childNodesAsProduction(node.getChildren)
      similarTrees ← uniqueProductionsToStubs.get(prod)
      similarTrees.add($tree_i$)
      vars ← uniqueProductionsToVariationsCounts.get(prod)
      uniqueProductionsToVariationCounts.set(vars + variations.get($tree_i$))
  {Recurse down through child nodes}
  for all $similarTrees_i$ ∈ uniqueProductionsToStubs.values do
      for all $n_j$ ∈ $similarTrees_i$.getFirst.getChildNodes do
          recurse($similarTrees_i$,$similarTrees_i$.getFirst.getNodeIndex($n_j$),variations,probs)
  {After recursion returns, update variation counts for each group}
  for all $prod_i$ ∈ uniqueProductionsToStubs.keys do
      newVars ← 0
      for all $stub_j$ ∈ uniqueProductionsToStubs.get($prod_i$) do
          newVars ← newVars + variations.get($stub_j$)
      uniqueProductionsToVariationCounts.put(prod,newVars)
  {Use the group with the most variations as the basis, find the smallest multiple k of this such that when calculating new
  variation counts for the other groups using the ratio of probabilities, each groups new variation count is a multiple of the
  total for that group}
  if uniqueProductionsToStubs.keys.size > 1 then
      largestProd ←findMostVariations(uniqueProductionsToVariationCounts)
      largestVars ← uniqueProductionsToVariationCounts.get(largestProd)
      largestProb ← probabilities.get(node.symbol).get(largestProd)
      k ← 1
      changed ← true
      while changed do
          changed ← false
          for all $prod_i$ ∈ uniqueProductionsToStubs.keys do
              if $prod_i$ = largestProd then
                  continue
              prob ← probabilities.get(node.symbol).get($prod_i$)
              newVars ← (k ∗ largestVars) ∗ (prob/largestProb)
              if newVars mod uniqueProductionsToVariationCounts.get(prod)= 0 then
                  changed ← true
                  k ← k + 1
                  break
      {Update the variations for each tree stub and return}
      for all $prod_i$ ∈ uniqueProductionsToStubs.keys do
          prob ← probabilities.get(node.symbol).get($prod_i$)
          newVars ← (k ∗ largestVars) ∗ (prob/largestProb)
          multiplier ← newVars/uniqueProductionsToVariationCounts.get($prod_i$)
          for all $stub_j$ ∈ uniqueProductionsToStubs.get($prod_i$) do
              variations.put($stub_j$,variations.get($stub_j$) ∗multiplier)
  return
```

Algorithm 8.3 Update variation counts on non-terminal elementary tree leaf nodes.

```
Require: $stubs$ {Elementary tree stubs}
Require: $variations$ {Updated map of elementary tree stubs to respective variation counts}
  for all $stub_i \in stub$ do
    $vars \leftarrow stub_i$.getVariations
    $multiplier \leftarrow variations$.get($stub_i$)$/vars$
    for all $n_j \in stub_i$.getNTLeafNodes do
      $n_j$.setVariations($n_j$.getVariations$*multiplier$)
```

and accompanying alternate hypothesis, H_{d1}, are stated:

H_{d0} : Imposing the CFG language biases on the TAGs generated from those CFGs has no effect on the performance of TAGE.

H_{d1} : Imposing the CFG language biases on the TAGs generated from those CFGs has an effect on the performance of TAGE.

α : 0.05 using the Mann-Whitney U, two-tailed, non-parametric test.

8.4.3 Results

The results of the comparisons of TAGE and PTAGE are presented in Table 8.7. It can be seen that PTAGE has a significant effect on the Santa Fe ant trail problem, as well as the six multiplexer problem, rejecting the null hypothesis, H_{d0}, for these problems. PTAGE has no effect on the remaining two problems.

8.4.4 Discussion

This section explores the effects of applying CFG biases to the TAGs generated from those CFGs. From Table 8.7 it can be seen that the application of PTAGE on two of the problems examined, Santa Fe and six multiplexer, has a significant effect on performance.

Table 8.7: A comparison of results obtained by TAGE when language biases from CFGs are applied to their equivalent TAGs. The mean best fitness values across 500 runs for the final generation, μ and the total number of successful runs are reported, correct to four decimal places. Mann-Whitney U test results comparing distributions of best fitness values at generation 200 are also presented. Shaded cells indicate p-values < 0.05. p-values for the comparisons are provided in Table C.5. PTAGE does not modify the grammars of EFP or SR.

	TAGE		PTAGE	
	μ (SD)	Successes	μ (SD)	Successes
EFP	0.0580 (0.5507)	493	0.0580 (0.5507)	493
SF	9.7080 (10.8598)	211	12.9840 (11.6440)	162
SR	0.0023 (0.0162)	485	0.0023 (0.0162)	485
6 Mux	0.5140 (1.6356)	442	2.5360 (2.9547)	251

PTAGE achieves considerably worse μ values that standard TAGE, and also finds less successful solutions. By examining the grammars of these two problems, it can be noted that, as a result of the transformation from CFG to TAG the production biases are altered, causing the selection of certain structures to be favoured over others. In the case of the six multiplexer problem, when selecting auxiliary trees for adjunction with TAGE there is a probability of ~ 0.81 of selecting an aux tree containing `<B>:<B>?<B>#`, a ~ 0.18 chance of selecting a tree containing either `<B> <B> |` or `<B> <B> &` each, and a < 0.01 chance of selecting a tree containing `<B> !`. This bias causes the TAGE tree to grow much wider than when using the PTAGE approach, as PTAGE balances these probabilities to be equal since they have an equal chance of being selected when deriving using CFGs. This is particularly evident when examining the depths and sizes of TAGE and PTAGE derived trees over the course of the runs in Figure 8.17. In this plot it can be seen that, on average, PTAGE trees are of a similar depth to TAGE trees, however, the tree sizes are much smaller. This indicates that PTAGE trees are narrow in relation to TAGE trees for this grammar. The stubs for the six multiplexer grammar are given in Figure 8.18. Similar effects are observed in the Santa Fe grammar. PTAGE has no effect on the other problems

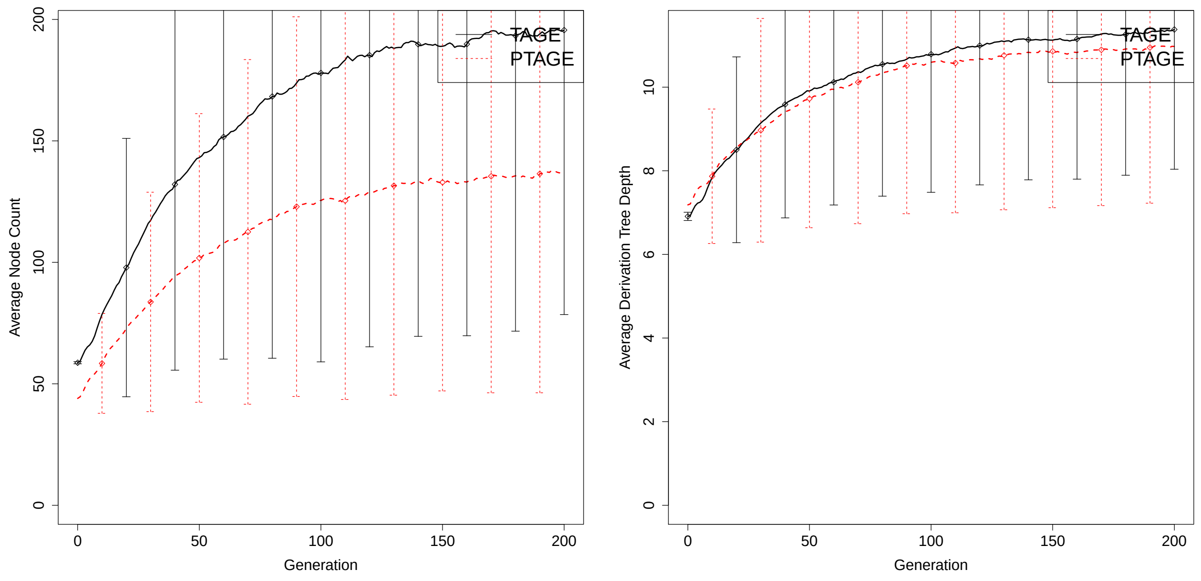


Figure 8.17: TAGE and PTAGE derived tree sizes and depths for the six multiplexer grammar.

as the grammar transformation does not introduce any new biases.

These examinations further highlight the fact that biases can be beneficial depending on the representation and the problem being addressed, in addition to showing how easily changed these biases are in TAGE.

8.5 Summary

This chapter investigates preferential language biases which are introduced when using TAGs in GE. These biases affect the distribution of generated derivation structures and resulting strings, both at initialisation and throughout an evolutionary run. The chapter also examines how common initial populations affect the comparison of TAGE and GE.

First, a separate comparison of GE and TAGE is performed, investigating whether the method of generating initial populations can be significant. This is shown to be true. As such, two methods of creating similar initial sets of trees for GE and TAGE are intro-

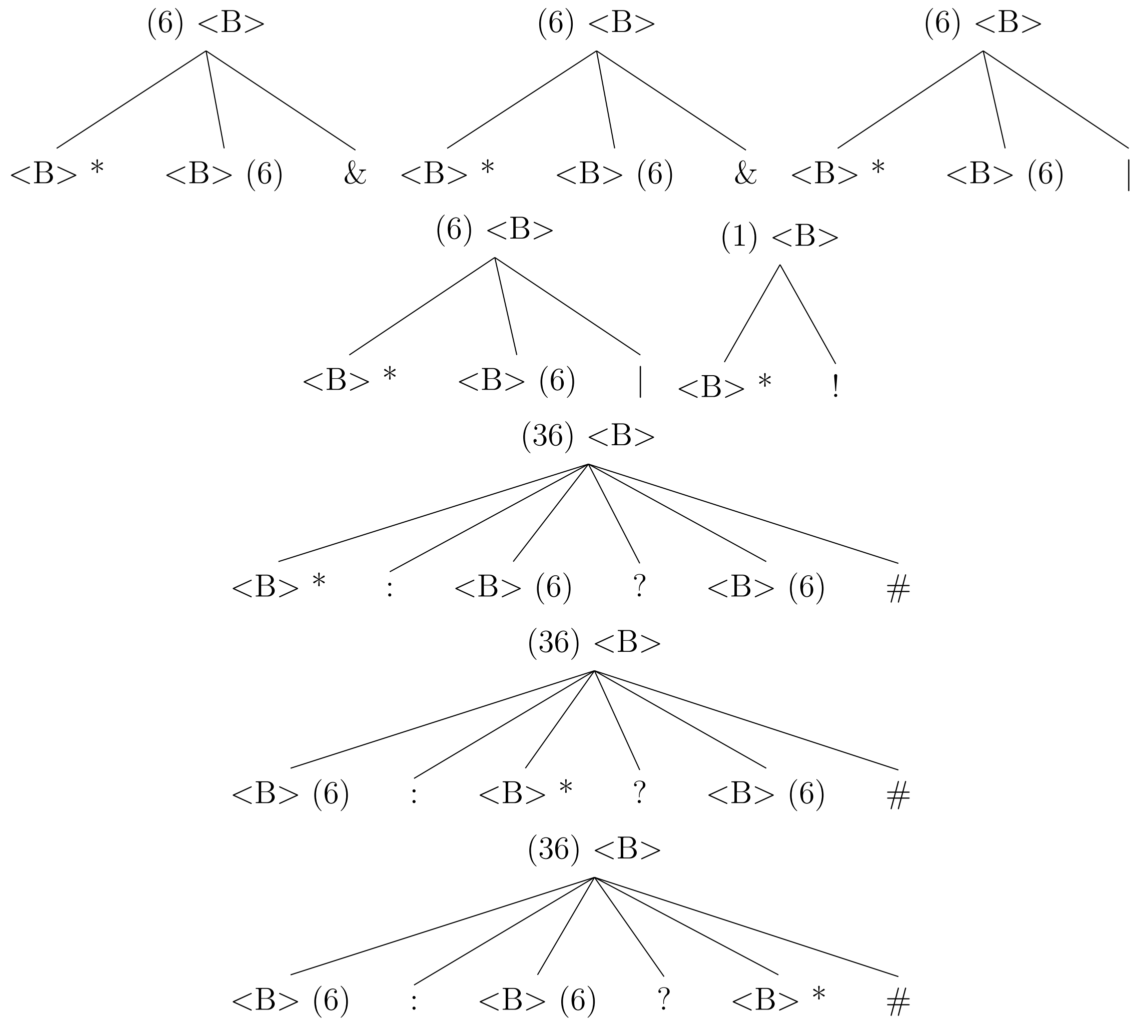


Figure 8.18: TAGE auxiliary tree stubs for the six multiplexer CFG in Figure 8.2. The total variations of each tree are written beside the root node of each tree. There is a total of 133 possible trees.

duced, and the performance of GE and TAGE is compared. This comparison shows that, even when initialised with identical populations of derivation trees and phenotypes, there remains a significant difference in performance in most of the problems examined, and of those significant differences, TAGE results in lower mean best fitness values and produces more successful solutions.

In generating similar populations of trees, biases in the distributions of the shapes of the trees being generated by TAGE are identified. These biases are introduced due to the constraints placed upon the adjunction operation by TAGE, as well as by the transformation algorithm used to generate TAGs from the original CFGs. New adjunction constraints and a system to eliminate the transformation biases, PTAGE, are described. An examination of the effect of these biases have on the performance of TAGE is presented and it is shown that these biases, adjunction restrictions and transformation biases, can be beneficial to the algorithm but are dependant on the grammar and the problem in question.

In the following chapter the developmental nature of the feasibility property of TAGs is explored, and a novel evolutionary developmental system is developed by integrating an online artificial gene regulatory network (GRN) model with TAGE.

Chapter 9

Developmental TAGE

GRNs, which are at the core of developmental biology allow for *"differential gene expression from the same nuclear repertoire"* [58]. These networks are capable of producing complex dynamics, partially achieved through the use of both direct and indirect self-regulation, as well as environmental factors. GRNs are attractive for EC as natural GRNs are known to be capable of organising as scale-free networks. Such networks have been shown to be modular in nature and robust to mutations, both important properties for evolvability. An outline of a new genetic representation for GP using GRNs is given by Banzhaf [5] (outlined in Section 4.2), exploring the dynamics of such a system. An extension of this model provides it with a means of input and output, allowing the model to be used as a computational tool [156]. However, even in its extended form, the model is limited in use as the phenotype produced is a series of real-valued signals, making the representation difficult to apply to many different problem domains.

This chapter is concerned with exploring the use of GRNs in conjunction with the generative power of TAGs, and investigating the effects of this complexification of the mapping process. TAGs are especially useful from a developmental perspective due to their property of feasibility. This means that all intermediate developmental stages are "functional", i.e., TAGs allow the production of valid individuals that can be evaluated at every stage of derivation.

The method presented in this thesis is novel as it introduces a combination of the dynamics present in the Banzhaf [5] model with the expressiveness of grammar formalisms. This method has the potential of producing different phenotypes - of any language that can be defined with a grammar - each time the GRN is sampled.

TAGE is extended to include the GRN model within the mapping process. Employing TAGs for the generation of phenotypes enables the GRN model to be applied to many different problem domains, while the GRN provides TAGE with rich dynamics for the generation of phenotypes, as well as enabling the problem environment to influence the mapping process. Some of the work in this chapter is based upon a study published by Murphy et al. [144].

The layout of the remainder of the chapter is as follows. The proceeding section, Section 9.1, outlines the integration of the extended Banzhaf-Nicolau GRN model (see Section 4.2) into the TAGE pipeline. This is tested on a common benchmark problem, the inverted pendulum problem, in Section 9.2. Limitations of the GRN-extended TAGE algorithm are discussed in Section 9.3. Finally, a summary of the chapter concludes the chapter in Section 9.4.

9.1 Developmental TAGE

This section introduces a novel extension of the GRN model described above. The model, as it stands, is capable of receiving signals from the environment as input, and providing signals back to the environment as output. However, these signals are limited to real values and, as such, it is difficult to apply to many different problem classes. In this section, a novel merging of the extended GRN model and TAGE is presented. This combination of methods, which will be referred to as Developmental TAGE (DTAGE), allows the power of grammar formalisms to be used in conjunction with the complex dynamics that the

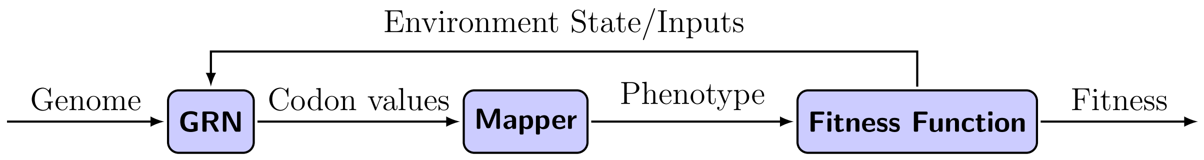


Figure 9.1: The path of an individual through the GRN-extended TAGE pipeline. The number of cycles is problem dependent.

GRN is capable of producing. The core concept underlying DTAGE is that of cell differentiation, i.e., differential gene expression. This involves the same genome giving rise to a number of different phenotypic effects as a result of factors external to the genome, be they environmental or physical in nature.

Taking dynamic environments an example, the aim of DTAGE is to evolve individuals such that a single individual is capable of expressing one of a variety of different phenotypes, depending on the current state of the environment. This is achieved by allowing the state of the environment to affect the genotype-phenotype mapping process.

In order to investigate the combination of the GRN model with grammar formalisms, the TAGE pipeline is extended. The GRN model is embedded within the mapping process. This enables easy access to the evolved genome in order to locate TF-genes and P-genes for the construction of the GRN. At a high level, the GRN takes state information from the problem environment and provides codons for the traditional TAGE mapping process. The generated phenotype affects the problem environment and new state information is passed to the GRN, which in turn provides new codons for mapping.

In order to allow for this, a feedback loop is required between the fitness function and the mapping process. This is highlighted in Figure 9.1, while Figure 9.2 shows the GRN in the context of the entire algorithm.

Example 3 (Evaluation in DTAGE) To evaluate an individual, its genome must first be mapped. The mapping call initiates a search of the individual's genome for genes and a GRN is constructed. This GRN is allowed to run, without free TF-proteins (inputs), for

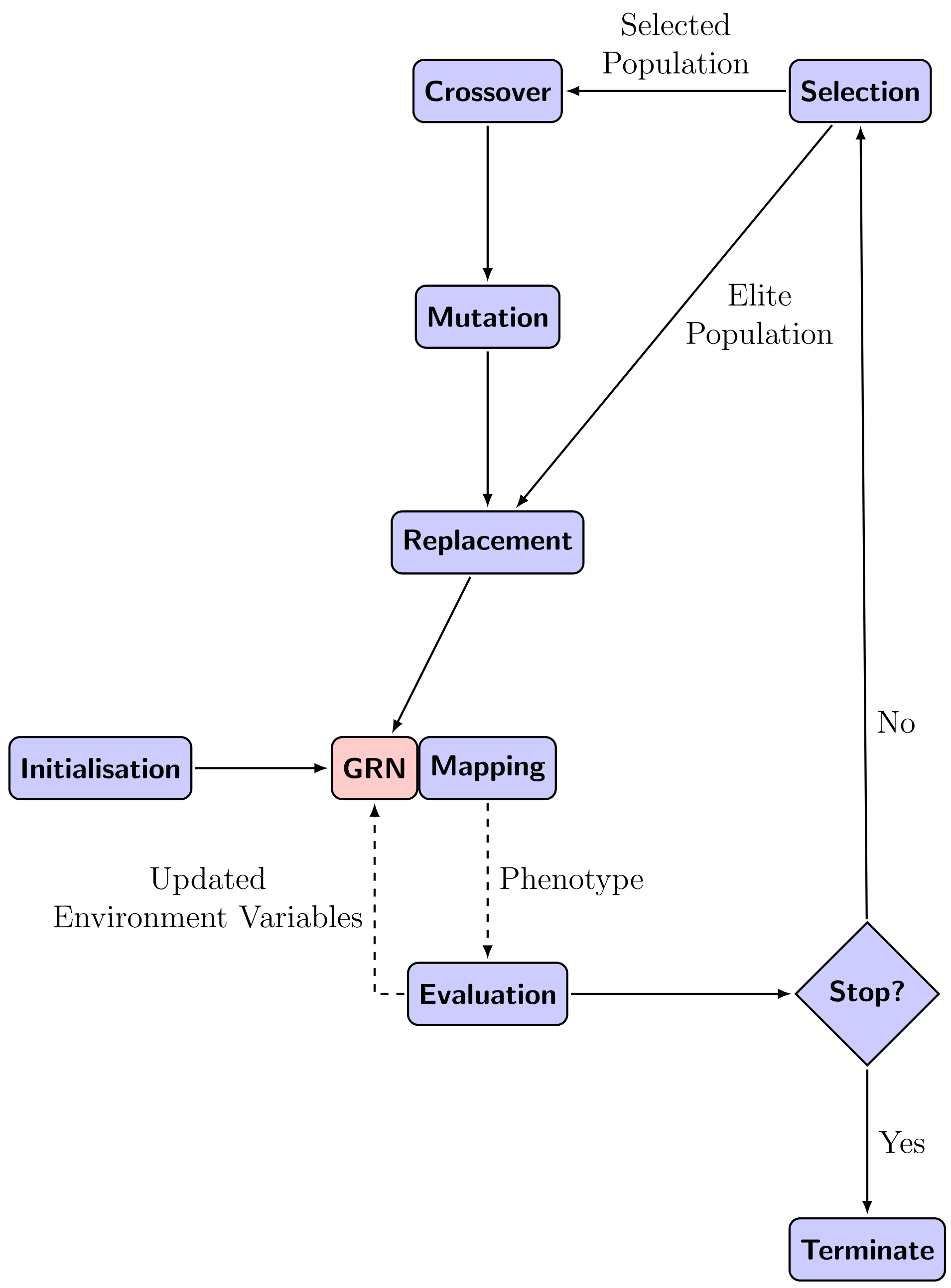


Figure 9.2: An updated flow graph of the GE algorithm (Figure 2.4) incorporating the GRN. Evolved genotypes are used to build GRNs, codons are extracted from the GRN model and mapped to a phenotype. Information from the evaluation environment can be fed back into the GRN, affecting the produced codons and resulting phenotype. This allows multiple phenotypes to be generated from individual genotypes.

a maximum of 10000 iterations in an effort to allow the network to stabilise into a steady state. This may stop prematurely if a steady state is detected in fewer iterations. The algorithm to detect if the GRN has stabilised is given in Algorithm 9.1.

Algorithm 9.1 Checks to see if the GRN has settled into a steady state, i.e., concentration values have stopped changing.

```
Require: $P$ the set of all protein concentration values over time
Require: $t$ the current iteration
Require: $T$ the size of window of iterations to check
Require: $\epsilon$ a threshold of change
  if $t < T$ then
    return false
  for all $p_i$ in $P$ do
    $\delta \leftarrow p_{it} - p_{i(t-T)}$
    if $|\delta| > \epsilon$ then
      return false
  return true
```

After stabilisation, the fitness function is called to evaluate the individual. The initial state of the environment, encoded as a set of free TF-protein concentrations, is passed to the mapper and injected into the GRN, scaling the existing produced TF-proteins as necessary. The injection of inputs and scaling can also happen before stabilisation so as not to unbalance the system.

The GRN is then run for a predefined number of iterations, *a sync*, before the P-proteins and their concentrations are used to create a string of integer codon values. These codons are then used by the traditional mapping process to produce a valid phenotype. This phenotype is evaluated by the fitness function and updated state information is injected into the GRN. The GRN is again allowed to sync, before extracting codons from the P-proteins and mapping a new phenotype to be passed to the fitness function. This process is repeated as many times as required by the fitness function. □

Using TAGs for this mapping is important as, unlike CFGs, TAGs will always produce a valid phenotype, regardless of how many codon values are available to the mapper. In addition, a broader range of phenotype sizes can be produced by TAGs than CFGs with the same number of codons. Equivalent phenotypes using CFGs would require much larger networks.

9.1.1 Protein-Phenotype Mapping

There are many different methods of interpreting the P-proteins and their concentration levels to produce codon values for mapping. In this thesis, four such methods are examined. The first two are for use with a very simple binary grammar, i.e., a TAG with only two initial trees, and are chosen to simulate the approaches taken by Nicolau et al. [156] while still retaining the use of a direct mapping grammar. These two methods make use of a single P-protein to produce a single codon value. The latter two methods are for use with grammars of any arity and use all available P-proteins to produce a chromosome of codon values, enabling mapping with grammars of much higher complexity.

Concentration Value This approach makes use of the concentration value of a single P-protein. If that value is found to be greater than 0.5, a codon value of `1` is used. Otherwise, a codon value of `0` is used. When using this approach, all possible P-proteins are tested, selecting the P-protein which generates the best fitness.

Concentration Tendency Here, the change in concentration of a single P-protein from input injection to the final iteration of a sync is used. A P-protein is considered if the magnitude of this change is greater than some *threshold*. A codon value of 0 or 1 is used depending on the sign of that change, positive or negative respectively. Otherwise, the codon value chosen previously will be reused. A threshold of 1^{-10} is used. Similarly, when using this approach all possible P-proteins are tested, selecting the P-protein which generates the best fitness.

Sort by Concentration This approach uses the 32 bit protein signatures of all P-proteins as codon values. The set of P-proteins is sorted in descending order by concentration level, as suggested by Banzhaf [5]. The 32 bit signatures of the P-proteins are then used to represent integer codon values, with the most concentrated P-protein as the left-most codon.

Sort by Concentration Tendency Similar to the *Sort by Concentration* method above, this approach uses the 32 bit protein signatures of all P-proteins as codon values. However, the P-proteins are sorted by the absolute magnitude of the change in their concentration values over the course of the sync. That is to say, the P-protein whose concentration changes the most over the course of a sync is used as the first codon for mapping, and so on, in decreasing magnitude of change until the P-protein whose concentration changed the least is used as the final codon in the sequence.

9.2 Benchmark

To test the ability of DTAGE to learn information about the problem environment and to produce suitable phenotypes to "survive" within the problem environment, the performance of DTAGE is observed on a standard dynamic benchmark problem, the inverted pendulum problem [9, 216]. This problem, also known as the pole-balancing or cart-pole problem, is a classic dynamic control problem that has been previously examined in the field of tree-based GP [110, 204, 205]. More recently, the problem has been examined in the study of the evolution of neural networks using Cartesian GP [106], and the of application of GRNs to evolutionary computation [156, 121]. The problem description below is based upon the problem setup used by Nicolau et al. [156].

9.2.1 Inverted Pendulum Problem

The problem consists of simulating a cart which can move along a finite two dimensional level track. A rigid pole is hinged to the centre of the cart. Forces may be applied to the cart in either direction, causing the cart and the pole to accelerate either left or right. The aim of the problem is to prevent the angle created between the pole and the vertical from becoming greater than some threshold, and keeping the cart within the bounds of the track. The problem model consists of four state variables:

$x \in [-2.4, +2.4]m$ the cart position, relative to the centre of the track;

$\dot{x} \in [-1.0, 1.0]m/s$ the velocity of the cart;

$\theta \in [-12, 12]^{\circ}$ the angle of the pole with the vertical;

$\dot{\theta} \in [-1.5, 1.5]^{\circ}/s$ the angular velocity of the pole.

The (friction-less) physical simulation of the cart-pole model is governed by the following non-linear differential equations:

$$\ddot{\theta}_t = \frac{g \sin\theta_t - \cos\theta_t \left[\frac{F_t + ml\dot{\theta}_t^2 \sin\theta_t}{m_c + m}\right]}{l\left[\frac{4}{3} - \frac{m\cos^2\theta_t}{m_c + m}\right]} \qquad \ddot{x}_t = \frac{F_t + ml\left[\dot{\theta}_t^2 \sin\theta_t - \ddot{\theta}_t \cos\theta_t\right]}{m_c + m},$$

where $g = -9.8m/s^2$, acceleration due to gravity, $m_c = 1,0kg$, the mass of cart, $m = 0.1kg$, the mass of pole, $l = 0.5m$, the half-pole length, $F_t = \alpha \cdot 10N$ where $\alpha \in [-1.0, 1.0]$, the force applied to the cart's center of mass at time t.

GRN Inputs

The four state variables mentioned above are encoded as free TF-proteins for the GRN model by assigning a unique signature to each variable, with each variable's value normalised in the range $[0.0, 0.1]$ taking up a maximum concentration of 0.4 in the GRN model. The signatures of each variable are:

x: `00000000000000000000000000000000` θ: `11111111111111111111111111111111`

$\dot{x}$: `11111111111111110000000000000000` $\dot{\theta}$: `00000000000000001111111111111111`

These free TF-proteins are injected into the stabilised GRN and the existing produced TF-proteins are scaled as required. The GRN is then synchronised using a sync step size of 2000 iterations. This sync step corresponds to $0.2s$ of simulated time for the cart-pole model. After which, the P-proteins are mapped and the phenotype is produced. This phenotype is evaluated resulting in a scaling co-efficient, α, for the force. The value of α is clamped between -1.0 and 1.0 inclusive. The scaled force is applied to the equations of motion and the state variables are updated, re-encoded, and fed back into the GRN. This process continues until the maximum number of time steps is reached, or the cart-pole model enters a failure state, i.e., $-2.4 > x > 2.4$ or $-12° > \theta > 12°$. Fitness is calculated by:

$$F(x) = \frac{120000}{successful\ time\ steps} - 1 \ .$$

9.2.2 Experimental Design

This chapter is concerned with the application of grammars to the artificial GRN model. The initial goal is to learn whether DTAGE is sufficient to capture state information from the problem environment and to evolve a mapping from different environmental states observed in the problem to some set of phenotypes which will enable individuals to successfully solve the problem. The secondary goals are to examine the effect of different methods of mapping from P-proteins to codon values, and to explore the use of different grammars, each defining a more complex language than the next.

To this end, several experiments are presented in the following sections. The grammars used throughout this study are listed as CFGs in Figure 9.3. These CFGs are converted to TAGs using Algorithm 3.1. The general evolutionary parameters of the system are presented in Table 9.1.

Table 9.1: GE parameters.

Parameter	Value
System	GEVA v1.1 (extended)
Random Number Generator	Mersenne Twister MT199973
Generations	50
Population Size	250
Initialisation Method 0	Random
Initial Chromosome Size	128 (4096 bits)
Max Chromosome Wraps	0
Replacement Strategy	Generational
Elitism	25 Individuals
Selection Operation	Tournament
Tournament Size	3
Crossover Probability	0.0
Bit Mutation Probability	0.005

In order to determine whether DTAGE is sufficient to solve the problem and is capable of performing comparably to similar work in the field, e.g., Nicolau et al. [156], 100 independent runs of the simple direct mapping grammar using the first two P-protein mapping methods (see Section 9.1.1) are performed. These methods generate a single codon value for mapping, producing a phenotype of either `-1` or `+1`. The problem environment is set to a random initial state at the start of each generation of a run. The entire population, including elites are evaluated using this new initial state and must balance the pole for 120,000 time steps (syncs) to successfully solve the problem. If at any generation an individual succeeds in solving the problem, then the run is stopped. Following this, to assess the effect of grammar complexity on the system, three different grammars of increasing complexity are examined over 100 runs using the remaining two P-protein mapping approaches.

In total, eight separate runs are performed and four comparisons are performed. First, the initial two mapping methods are compared using the direct mapping grammar. Following this, each of the more complex grammars are used to compare the remaining two mapping methods.

```
<power>  ::= 1.0 0.0 - | 1.0
```

(a) Direct mapping grammar

```
<power>  ::= <const> 0.0 <op>
<op>     ::= + | -
<const> ::= 0.0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
            0.6 | 0.7 | 0.8 | 0.9 | 1.0
```

(b) Discrete digits grammar

```
<power>  ::= <const> 0.0 <op>
<op>     ::= + | -
<const> ::= 0.<digits>
<digits> ::= <digit><digits> | <digit>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 |
            6 | 7 | 8 | 9
```

(c) Continuous digits grammar

```
<power> ::= <expr> <expr> <op>
<expr>  ::= <expr> <expr> <op> | <const>
<op>    ::= + | - | * | /
<const> ::= 0.<digit>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 |
            6 | 7 | 8 | 9
```

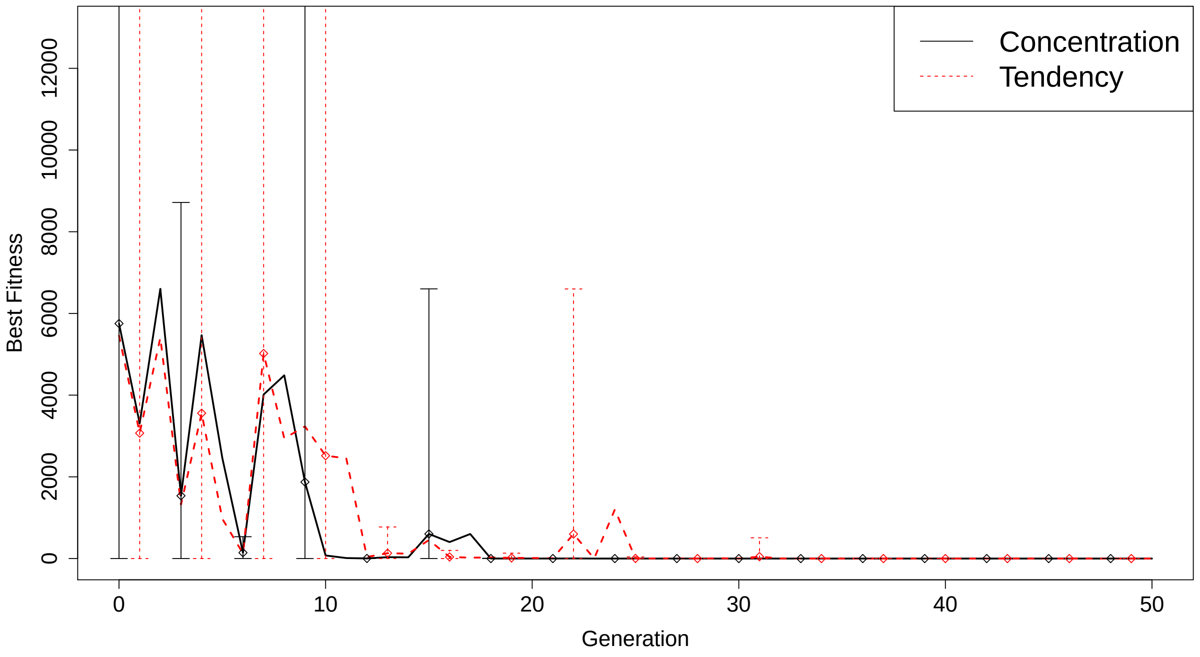
(d) Symbolic regression grammar

Figure 9.3: The grammars used, in reverse polish notation. Note that (c) generates the interval $(-1.0, 1.0)$. All resulting values are clamped to $[-1.0, 1.0]$. TAGs generated from these CFGs are given in Figure D.1-D.4.

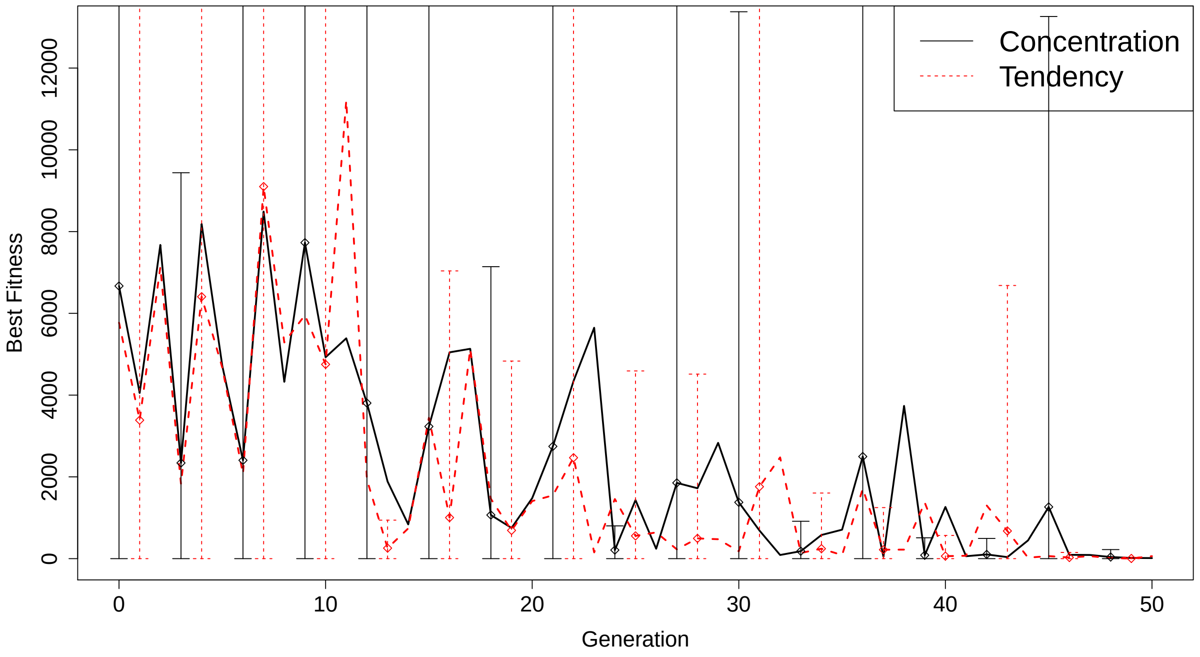
9.2.3 Results and Discussion

The mean best fitness plot of the direct mapping grammar experiments are presented in Figure 9.4a. From this plot, it appears that both approaches perform well on the problem. From the number of successful runs reported in Table 9.2 it can be seen that both mapping methods, using the concentration values and concentration tendencies manage to solve all runs. It is clear that the GRN model embedded in DTAGE representation is capable of being evolved to capture state information from the problem environment and generating the correct output to solve inverted pendulum problem. Similar plots are presented for the three other grammars, the discrete grammar in Figure 9.4b, continuous in Figure 9.5a and the symbolic regression grammar in Figure 9.5b.

Due to the dynamic nature of the problem, which allows for large variance between generations it is difficult to draw conclusions from these plots, as to which, if any, method of mapping from P-proteins performs better than another. As such, the results of the generalisation test are presented below, and help provide a clearer picture.

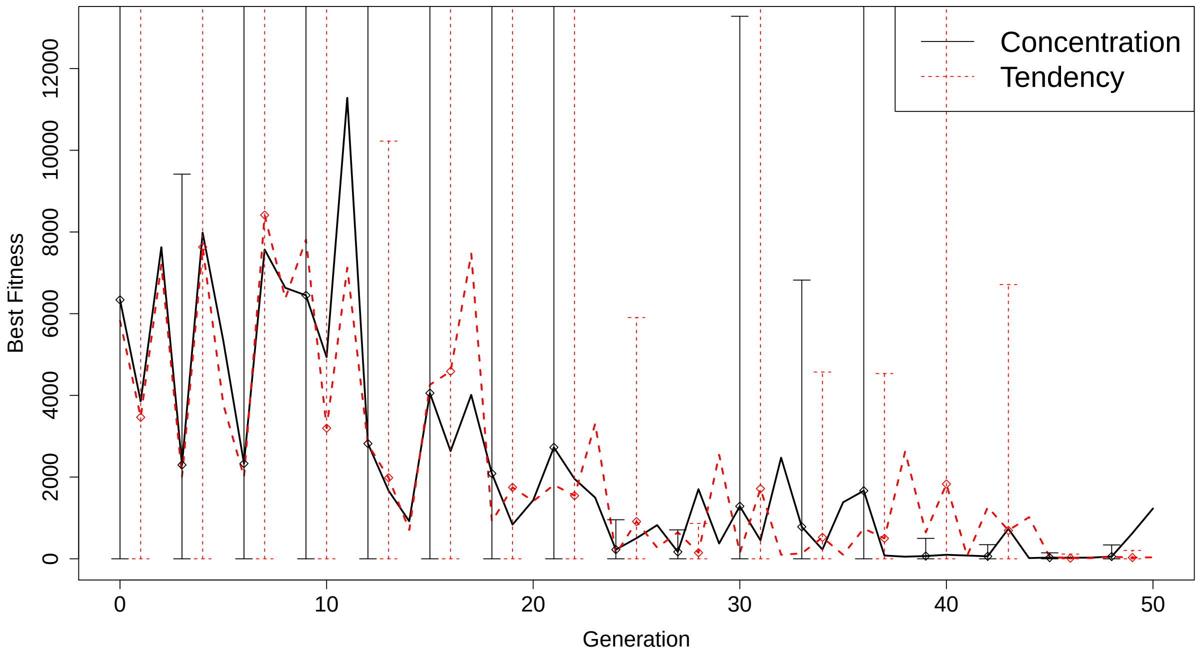


(a) Direct mapping grammar

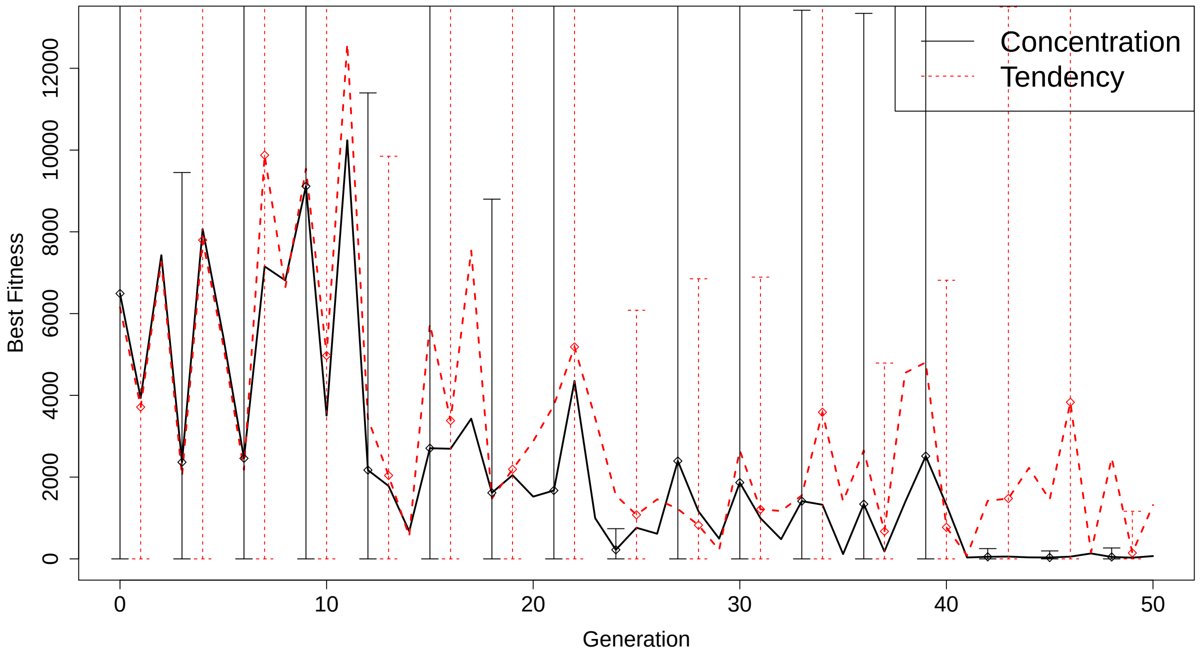


(b) Discrete grammar

Figure 9.4: Mean best fitness plots for the direct and discrete grammars comparing the use of concentration value against concentration tendency for choosing codons for mapping.



(a) Continuous grammar



(b) Symbolic regression grammar

Figure 9.5: Mean best fitness plots for the continuous and symbolic regression grammars comparing the use of concentration value against concentration tendency for choosing codons for mapping.

Table 9.2: Generalisation test results: 1000 time steps for 625 test cases. The number of successful runs are also reported.

Approach	Best	Worst	Median	Mean	Std. Dev.	Suc. (100)
Direct - Concentration	404	0	206.5	206.79	117.25	100
Direct - Tendency	232	1	64	75.08	51.08	100
Discrete - Concentration	409	0	196	185.77	106.93	94
Discrete - Tendency	268	0	110	104.37	56.72	95
Continuous - Concentration	393	0	198.5	184.58	108.96	92
Continuous - Tendency	241	0	109	108.1	53.28	94
Sym. Reg. - Concentration	393	0	167	158.35	109.25	91
Sym. Reg. - Tendency	321	0	106.5	109.31	68.85	82

Generalisation Results

The generalisation test proposed by Whitley et al. [216] is used in this study to examine robustness, i.e., how well a solution manages to survive when placed into a number of differing environmental situations. After each run, the best individual is tested on its ability to balance for 1000 time steps on 625 different test cases (start states). For each of the cart model's four state variables, a set of five values is calculated, with these values normalised to 0.05, 0.275, 0.725 and 0.95 of the each variable's range. The set of test cases is calculated from all combinations of values for each state variable. It should be noted that Nicolau et al. [156] have shown that only 457 of these cases are solvable. Similar generalisation tests for this problem have been used in the literature, e.g., Khan et al. [106] report generalisation success rates of 532 out of 625 test cases, however, these test cases are drawn at random; and Shimooka and Fujimoto [204] calculate robustness as the success rate over 1000 iterations with the test cases drawn from specified normal distributions.

Table 9.2 outlines the different results of the generalisations tests applied to the best individual of each run for each experiment. From this table it can be seen that there is a clear advantage to using the concentration values themselves over the tendency of these values to make decision for mapping. This may be due to the fact that the concentration values are related by means of their values being normalised, if one concentration decreases at least one other must increase. Such a relationship may be easier to evolve to solve this problem. Whereas with such a large sync size (2000 iterations), evolving the network in such a way as to produce the desired mapping from input states to the correct ordering of P-proteins is very difficult due to the complex dynamics present in the model. This difficulty increases as the size of the sync increases.

With regards to the different grammars examined, the results of the generalisation tests on the basic direct mapping grammar are comparable to those presented previously in the literature by Nicolau et al. [156] using random genomes, as is the case here. Similar to what is observed in this study for these results when comparing the usage of concentration value against concentration tendency, Nicolau et al. reports a reduction in performance when using tendency over the concentration value itself.

The introduction of larger, more expressive grammars is shown to affect performance. Keeping with the concentration approach, the mean test results for the discrete and continuous grammars decrease by $\sim$20% from the direct mapping approach. This decreases further for the symbolic regression grammar. It should be noted however, that while the means have dropped, the best result found over all the runs makes use of the discrete constant grammar.

From these results, it is clear that with the addition of the TAGE mapping process, the GRN model remains capable of being evolved to learn useful state information from the problem environment and to produce usable output. DTAGE enables the GRN model to be applied to many more interesting problem domains.

9.3 Limitations

The main limitation of using this GRN model is the time cost associated with it. Each time-step of the problem environment that the GRN is synchronised with requires, in this case, 2000 iterations of the GRN. In addition, the size of the networks in use in this study are relatively small, tens of genes at most. As the regulation of the Banzhaf model is O(n^2), increasing the size of the networks would compound this problem. The model includes scaling parameters which when used can increase the size of the regulatory effect. Increasing this effect can reduce the accuracy of the model but in theory this could allow for similar performance using a smaller sync size.

9.4 Summary

The objective of this chapter is to explore the effect of a novel combination of TAGs and an artificial GRN model. GRNs are thought to be core to the process of differential gene expression, and as such are of interest to the field of EC.

By exploiting the generative power of grammars, this GRN model can be further extended to allow more varied phenotypic products, enabling the application of the GRN model to a broader range of problem domains. Addressing this, the artificial GRN model is adapted into the TAGE algorithm. TAGs are ideal for this due to their property of feasibility, and larger base derivation structures. Different methods of extracting output from the GRN model for mapping using a TAG are examined. This new system is then tested on the inverted pendulum benchmark problem.

The results obtained show that the inclusion of a grammar is not detrimental to the performance of the GRN in the simplest case, reproducing results previously published in the literature [156], with more complex grammars highlighting the system's ability to produce and operate effectively with a more complex genotype-phenotype mapping. Of the two methods of mapping from GRN proteins to phenotypes that are tested, it is shown that using the concentration values to obtain codons for mapping results in improved performance over using the change in concentration.

9.4. SUMMARY

The end of this chapter brings a close to the exploration of TAGs as part of a representation for EC, and GE in particular. The following chapter presents a discussion of the conclusions of this thesis. Limitations of the work presented throughout are outlined, and opportunities for interesting future work which may be pursued on the use of TAGs for GE are presented.

Part III

Conclusions

Chapter 10

Conclusions and Future Work

Having explored the use of Tree-Adjoining Grammars for Grammatical Evolution, a discussion of the conclusions of this thesis is now presented. First, a thesis summary, along with the contributions it has made, is given in Section 10.1. The limitations of this thesis are outlined in Section 10.2. Finally, opportunities for future work which may be pursued on the use of TAGs for GE are given in Section 10.3.

10.1 Summary of Thesis

The aim of this thesis was to explore the use of TAGs as part of the representation for GE, and to investigate how the use of TAGs, which have been shown to be more powerful than CFGs, affects GE's ability to perform search. To fulfil this aim a number of research questions were proposed:

- *Can TAGs be used, in conjunction with a linear genotype, as an effective representation for GE?*
- *How does a TAG-based GE representation behave differently than the canonical CFG representation used by GE?*
- *Do the biases introduced by converting CFGs to TAGs affect GE?*

- *Can any properties of TAGs be exploited in order to enhance GE?*

Initially, Chapter 5 described a novel extension of GE incorporating TAGs into the GE representation (TAGE). Following this, Chapter 6 compared TAGE and standard GE in terms of performance. Differences in the two representations were identified and further comparisons were performed. Results from Chapter 6 initially show that TAGE outperforms GE on all problems examined. However, when the representational differences identified in this chapter were addressed, GE performed as well as, and indeed better than TAGE on some of the experiments. The results additionally highlight the fact that TAGE populations tend to maintain an average fitness much farther away from the optimum than GE.

Next, Chapter 7 explored the difference in connectivity of the search spaces of TAGE and GE using a novel method of rendering search landscapes. It was shown that, for single-mutation landscapes for a number of typical grammars, TAGE search spaces are much more densely connected than those of GE. This affords individuals greater opportunity to move about the space, which might aid in deterring population convergence. It can also indicate lower locality in the representation, allowing small changes to the genotype to have a wide range of effects on fitness.

Chapter 7 examined the effects of common phenotype initialisation and changes in the biases inherent to the representation when transforming from CFG to TAG. While the original CFG and the produced TAG are equivalent, i.e., no exclusion biases are introduced during the transformation, the preferential language biases in the system are affected by this transformation. Methods for addressing the different biases were given. The results from this chapter show that the biases that are induced can be beneficial to the representation, however, this benefit is shown to be problem dependent.

The final experimental chapter, Chapter 9, introduced DTAGE, a novel evolutionary developmental extension of TAGE using an artificial GRN model which exploits the feasibility feature of TAGs. The ability of this model to learn useful state information from the problem environment and to produce phenotypes to survive within the environment was tested on a common benchmark problem. The results of this chapter show that DTAGE is capable of evolving GRNs which can capture state information from a dynamical environment, and which can be used to generate codon sequences for mapping to phenotypes generated from complex grammars. The results of this chapter suggest that the extension might perform well on other more complex dynamic problems.

10.1.1 Contributions

A review of the literature on the use of TAGs in EC was provided in Chapter 3. Aside from this, the following contributions, and how they address the research questions of this thesis, are presented. Additional practical contributions resulting from the work completed are also provided.

TAGs as an effective representation

GE, and indeed many GBGP systems make use of CFGs as part of their representation. This thesis is interested in exploring the use of the more powerful TAGs for use in GE. In addressing Question 1, Chapter 5 presents a novel extension of GE which makes use of TAGs. In order to guarantee both the *non-fixed arity*, and the resulting *feasibility* properties of TAGs for use with GE, the adjunction composition operation is used exclusively.

Analysis of TAGE performance

Section 6.1 further contributes towards Question 1. The results from the experiments carried out in this section show that a TAG based GE representation (TAGE) performs better than the canonical GE. Confirming that TAGs can be used as part of an effective representation for GE.

Representational Differences

Question 2 is addressed by Section 6.2. Differences between the two representations are identified and when addressed, GE can perform as well, if not better than TAGE on some of the experiments. These investigations show that TAGE and GE behave differently.

Population Convergence

Question 2 is further addressed by the analysis of the average population fitness throughout the experiments performed in Chapter 6. The results in this chapter show that, while performing better in terms of best fitness, TAGE populations maintain worse average fitness than those of canonical GE, i.e, the population does not converge towards the optimum.

Connectivity of mutation landscapes

Moreover, Chapter 7, in examining the mutation search spaces for both TAGE and GE further contributes towards Question 2. Using a visualisation method on the mutation search landscapes, it is shown that the TAGE representation is much more densely connected than GE, affording TAGE individuals greater opportunities to traverse more of the search space. This can be indicative of a greater ability to escape from local fitness optima.

Analysis of language biases

In comparing the random generation of individuals, both for use with initialisation and variation throughout a run, Chapter 8 broaches Question 3. Section 8.3 explores the effects of adjunction constraints, while Section 8.4 investigates the modification of implicit biases in CFGs when transformed into TAGs. The results of these examinations show that the biases that are induced can be beneficial to the representation, however the benefit is shown to be problem dependent.

Beneficial TAG properties for GE

Throughout the work in this thesis, a number of properties of TAGs have been highlighted that address Question 4. As a result of the TAG representation developed in Chapter 5, the non-fixed arity and resulting feasibility properties of TAGs enable the elimination of invalid individuals from the GE system. This is achieved without the use of expensive repair operations. Furthermore, the enhanced connectivity of the TAGE representation shown in Chapter 7 - previously mentioned - provides benefit to the algorithm.

Applicability of GRNs

Moreover, further addressing Question 4, this property of feasibility enables the enhancement of GE by means of the incorporation of GRNs into TAGE, as shown in Section 9.1. The existing form of the GRN model is slow to execute and, as such, smaller GRNs are more desirable for practical uses. TAGs enables very few codons - generated from the GRN - to be used to create a large set of potential phenotypes.

The occurrence of invalids when mapping with CFGs makes this non-trivial. In order to make use of CFGs, a very simple grammar would be required to increase the likelihood of a valid mapping. Alternatively, wrapping or a method of segmenting the existing codons into additional codons could be used. There is no guarantee, however, with wrapping that mapping will terminate [157].

The incorporation of GRNs allows TAG-based GE to be applied in a novel way to a new class of problems. This extension of TAGE tested a number of different grammars and mapping procedures and it is shown that this complexification of the mapping process does not detract from the ability to find solutions.

The most significant of these contributions is the development of the TAGE representation as an effective representation. The incorporation of TAGs into the GE representation enables GE to make use of the expressive power of TAGs. In addition to allowing GE to take advantage of the grammar's properties, opening up possible avenues of exploration such as incremental evaluation and other forms of developmental evolutionary computation, such as presented with DTAGE.

Additional Practical Contributions

Development and implementation of TAG-based GE system

An open source GE framework, GEVA, is extended to make use of the TAG-based representation, TAGE, described in Chapter 5.

Method of initialisation for identical initial GE and TAGE populations

When comparing different representations, it can be difficult to account for the initial position in the search space. Section 8.2 presents a method of generating well distributed initial populations which can be used by both GE and TAGE by means of a shared intermediate state of representation. This allows a population of individuals to be generated in a normal manner for TAGE, generating the GE chromosomes for the same derivation trees and phenotypes retrospectively.

Method of imposing language biases with TAGs

The specific stub-based TAGE representation, presented in Section 5.4, is shown to be useful by allowing trivial modification of the language biases in a system.

Method of applying CFG biases to TAGs

The stub-based representation mentioned above is used as part of an algorithm to impose the probabilities of production rules being selected in CFGs on the TAGs produced by those CFGs. This method, PTAGE, is presented in Section 8.4.

Development of a novel developmental extension of TAGE

A developmental extension of TAGE where an artificial GRN model is incorporated into the TAGE pipeline is presented in Section 9.1. This extension, which augments the mapping process with a GRN, allows the complex dynamics that come about from this model to be taken advantage of by TAGE. In addition, the GRN model can now be utilised on the diverse set of problem classes that grammar-based GP are applied to. Moreover, this extension of the TAGE mapping process enables it to be influenced by the problem state, or environment, providing individuals with the ability to express different phenotypic effects depending on the environment in which they find themselves.

10.2 Limitations

As was mentioned in Chapter 1, the aim of this thesis is to explore the use of TAGs as part of a representation for GE. To this end, a novel representation is developed, some limitations must be considered to progress the study.

When developing a new representation decisions must be made on how best to achieve that goal. This thesis concentrates on a single developed representation, TAGE, and it's extensions. While many other methods of incorporating TAGs into the GE representation may exist, they have not been explored by in work.

Additionally, when examining this novel representation, only TAGs which have been produced from the CFG used by GE are used. This is done to ensure equivalence of the grammars, i.e., that they are capable of generating the same set of sentences and trees. No investigation into the design of custom TAGs for use with TAGE is provided.

Continuing with grammars, the CFGs used throughout are also the standard CFGs commonly used with GE on the problems examined. Recent work has looked at how CFGs can be written to alleviate certain biases, and properties which could improve standard GE's performance [63, 157]. Furthermore, no comparison is provided of TAGE with other grammar types that have been used with GE, e.g., attribute grammars.

As with all EAs, there a number of decisions which must be made in order to set evolution in motion. These decisions include, the choice of operators, as well as the setting of rates for mutation, crossover, selection and replacement. The parameters used in the investigations within this work are set as being similar to the standard values used in the field. No exhaustive parameter sweep of these values or operator combinations is performed. Similarly, not all possible problems can be examined so a representative set of benchmark problems from the literature are examined.

This limitation of parameter selection extends to the use of the GRN model described in Section 4.2. No exploration of network size is performed, nor is there an investigation into the length of each synchronisation period or the scaling terms. The values used are taken from the literature.

Finally, there exists a large body of literature on evolutionary developmental systems which could be examined to help improve the DTAGE extension.

10.3 Opportunities for Future Research

The work presented in this thesis has shown that TAGs can be effectively used as part of the representation for GE. Through the completion of this exploration a number of interesting avenues for future work have been uncovered for exploitation. This section provides an outline of some of these interesting opportunities, in order of priority in the opinion of the author.

Diversity

As is shown in Chapter 7, TAGE landscapes for mutation operations appear far more densely connected than those of GE, and in addition, shown in Chapter 6, TAGE populations show a trend of non-convergence towards the optimum. An investigation should be carried out to ascertain whether these properties of TAGE aid to enhance, or indeed affect, the diversity of populations generated by TAGE at each generation.

Substitution Composition Operation

To ensure feasibility in the developed TAG-based representation, the substitution composition operation is omitted, and the adjunction operation is exclusively used. Substitution allows for a much more compact representation and is a desirable property when utilising TAGs. This works presents a single method of incorporating TAGs with GE. Further study could examine other methods of incorporating TAGs which might enable the use of substitution. Such as adopting the lexeme approach described in Section 3.2 for TAGE. Such an approach could possibly be utilised by means of a hybrid representation, making use of the linear chromosome of GE to evolve adjunction operations and separately making use of safe operations which modify substitution composition operations while ensuring each tree is complete. Another possibility would be wrap the chromosome which is used for substitution in a diploid approach, ensuring mapping completion. Alternatively, further investigation into the presented TAGE representation may reveal a method which enables the use the substitution composition operation.

As a result of this design decision, the tree stub-based approach is used. While helping to reduce the number of trees which need to be generated, it does not alleviate the problem completely. CFGs, which, rather than having rules with large numbers of terminal productions, have a large number of non-terminal productions, are not addressed by the stub approach. If substitution cannot be incorporated into TAGE, further work on this method of tree set reduction could be carried out to improve the representation.

Variation Operations

The examination of the mutation search landscapes revealed that the TAGE representation is much more densely connected. This study opens up a number of further avenues of interesting work. First, the novel method analysis used for the mutation landscapes could be extended to other operations, such as crossover. In doing so, the dimensionaltiy of the visualisation will increase, making it a difficult but interesting problem. The application of these methods does not stop with GE and TAGE, and could be applied to the analysis of other types of EA.

Secondly, while the TAGE representation is shown to be more dense, with greater opportunities to move from each point in the search space, by recalling how TAGE mapping operates and that TAGs have the non-fixed arity property, an interesting suggestion arises that could warrant further investigation. This suggestion is that TAGE mutation may be capable of providing higher structural locality than mutation in standard GE. If a mutation event does not modify the number of adjoinable address that would have been introduced into the tree should the mutation not have happened, then a mutation event with a very high structural locality will have occurred, with only terminal symbols throughout the tree being affected, and the structure of the tree remaining the same.

Furthermore, there are many opportunities for further exploration of TAGE in the context of variation operations. Only the basic GA crossover and mutation operations have been used throughout this thesis. Investigation into other interesting operations, such as a less destructive sub-tree crossover, could be conducted using TAGE.

DTAGE

Further work can be conducted into exploring the impact of network size, as well as the length of synchronisation period. Such studies would be advantageous, allowing application of the model to more complex dynamic problems with an increased number of environmental variables.

In addition, much interesting study remains to be performed on analysing the phenotypes produced by the DTAGE mapping combination, as well as examining how the different methods of extracting codons from the GRN affects this.

TAG exclusive languages

As TAGs is slightly more powerful than CFGs, there are languages which TAGs can generate and which CFGs cannot. An investigation should be performed to examine whether these types of languages can provide any benefit to TAGE.

TAG derivation tree shape

Tree balancing algorithms exist and could be made use of for the generation of TAG derivation trees for TAGE initialisation. An investigation should be performed to examine the suitability of such algorithms to generate desirable distributions of derivation tree shapes and sizes.

Bloat

The final opportunities for future work presented here are those pertaining to Bloat. Bloat is a concern for any representation that generates large trees of code. As there is no bias in TAGs towards specific derivation tree sizes, and composition operations can operate on large groups of symbols (elementary trees) than GE, resulting in larger trees on average than GE for the same chromosome length, an investigative study on the level of bloat would be of benefit to TAGE.

Appendix A

Chapter 3

```
<prog> ::= <setup><main>

# Mandatory setup
<setup> ::= \t\tthresholdDistanceGhosts = <ghostThreshold>;
            \n\t\twindowSize = <window>;
            \n\t\tavoidGhostDistance = <avoidDistance>;
            \n\t\tavgDistBetGhosts = (int)adbg.score(gs,thresholdDistanceGhosts);
            \n\t\tang.score(gs, current, windowSize);\n

# We always want these conditions
<main> ::= if(edibleGhosts == 0){\n \t<statements>\n }\n
           else {\n \t<statements>\n }\n

# This part is recurisve
<statements> ::= if( <condition> ) {\n <action> }\n <elses>
               | if( <condition> ) {\n <statements>\n }\n <elses>
               | if( avgDistBetGhosts <lessX2> thresholdDistanceGhosts ) {\n <actsOrStats>\n }\n <elses>
               | if(inedibleGhostDistance <lessX2> windowSize) {\n <avoidOrPPill> }\n <elses>

<elses> ::= else {\n <action> }\n |  else {\n <statements>\n }\n | else {}\n

<actsOrStats> ::= <action> | <statements>
<action> ::= \tnext = Utilities.getClosest(current.adj, <closest>, gs.getMaze());\n
           | if (numPowerPills <more> 0){\n\t<pPillAction> }\n
             else{\n\tnext = Utilities.getClosest(current.adj, npd.closest, gs.getMaze());\n}\n

<closest> ::= npd.closest | ang.closest | ngd.closest
<avoidOrPPill> ::= <avoidAction> | <pPillAction>
<avoidAction> ::= \tnext = Utilities.getClosest(current.adj, <avoidClosest>, gs.getMaze());\n
<pPillAction> ::= \tnext = Utilities.getClosest(current.adj, <pPillClosest>, gs.getMaze());\n
<avoidClosest> ::= ang.closest
<pPillClosest> ::= nppd.closest

# Variables
<condition> ::= <var> <comparison> <var>
<var> ::= thresholdDistanceGhosts | inedibleGhostDistance
        | avgDistBetGhosts | avoidGhostDistance | windowSize

<ghostThreshold> ::=  1 |  2 |  3 |  4 |  5 |  6 |  7 |  8 |  9 | 10
                   | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20
<avoidDistance> ::=   1 |  2 |  3 |  4 |  5 |  6 |  7 |  8 |  9 | 10
                   | 11 | 12 | 13 | 14 | 15
<window> ::= 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19

# Inequality comparisons
<comparison> ::= <less> | <more> | <lessE> | <moreE> | <equals>
<lessX2> ::= <less> | <lessE>
<less> ::= "<"
<more> ::= ">"
<lessE> ::= "<="
<moreE> ::= ">="
<equals> ::= "=="
```

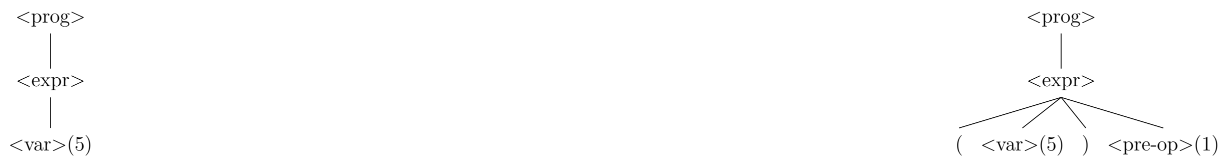
Figure A.1: Example of a complex grammar as referenced in Section 5.4. This grammar was used for the Mrs. Pacman problem by Galvan-Lopez et al. [54].

Appendix B

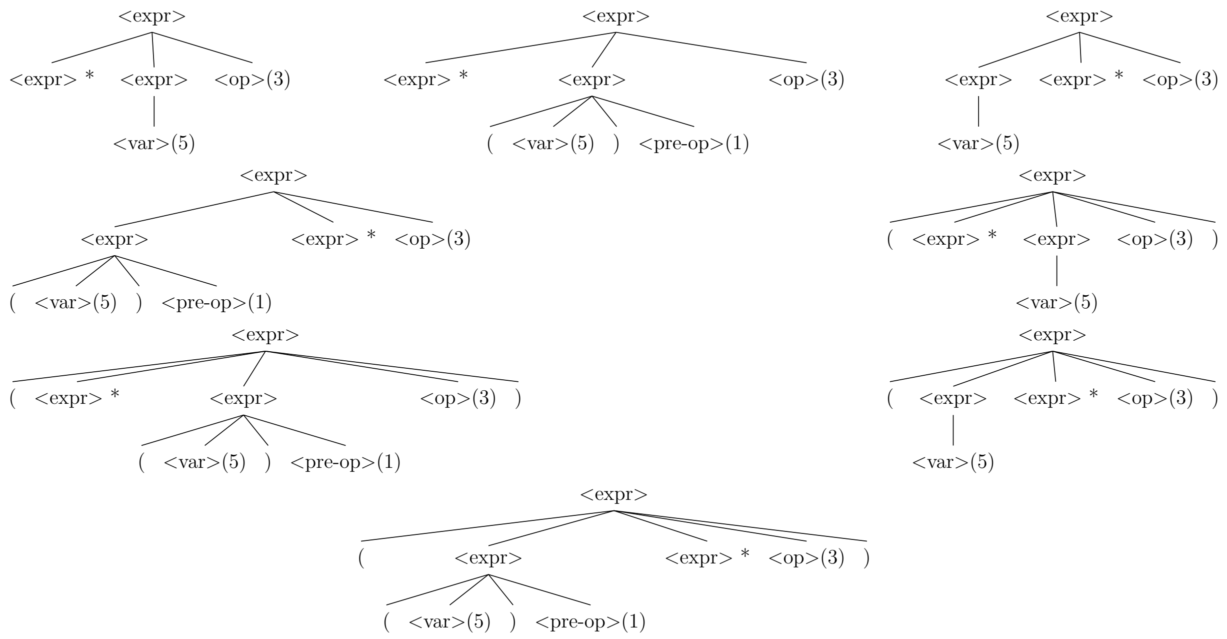
Chapter 5 & Chapter 6

```
<prog>   ::= <expr>
<expr>   ::= <expr> <expr> <op>
           | ( <expr> <expr>  <op> )
           | <var>
           | ( <var> ) <pre-op>
<pre-op> ::= !
<op>     ::= "|" | & | ^
<var>    ::= d0 | d1 | d2 | d3 | d4
```

(a) CFG



(b) Initial trees

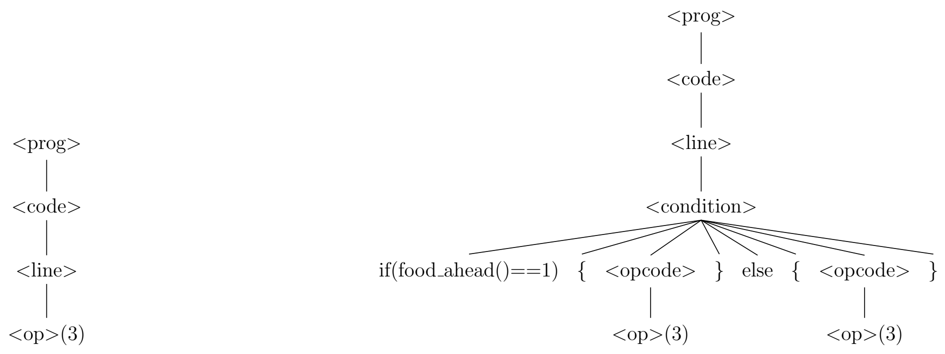


(c) Auxiliary trees

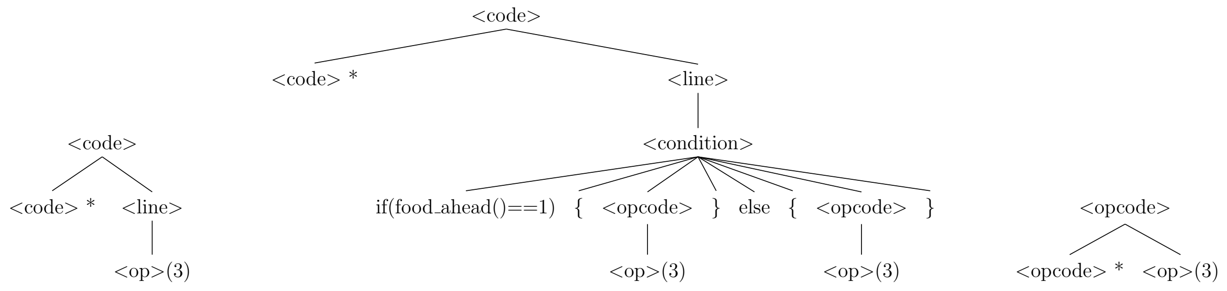
Figure B.1: CFG and TAG for the even-five parity problem. The number in parenthesis indicates the amount of different variations of terminal productions that can be attached at that node.

```
<prog>      ::= <code>
<code>      ::= <line> | <code> <line>
<line>      ::= <condition> | <op>
<condition> ::= if(food_ahead()==1) { <opcode> }
                else { <opcode> }
<op>        ::= left(); | right(); | move();
<opcode>    ::= <op> | <opcode> <op>
```

(a) CFG



(b) Initial trees

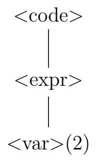


(c) Auxiliary trees

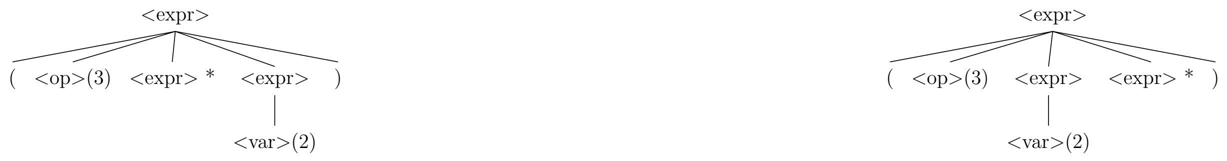
Figure B.2: CFG and TAG for the Santa Fe ant trail problem. The number in parenthesis indicates the amount of different variations of terminal productions that can be attached at that node.

```
<prog> ::= <expr>
<expr> ::= ( <op> <expr> <expr> ) | <var>
<op>   ::= + | - | *
<var>  ::= x0 | 1.0
```

(a) CFG



(b) Initial trees

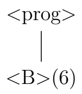


(c) Auxiliary trees

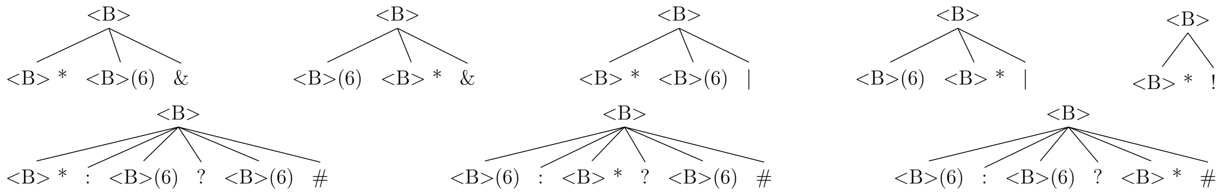
Figure B.3: CFG and TAG for the symbolic regression problem. The number in parenthesis indicates the amount of different variations of terminal productions that can be attached at that node.

```
<prog> ::= <B>
<B>    ::= <B> <B> & | <B> <B> "|"
         | <B> !
         | <B> : <B> ? <B> #
         | a0 | a1 | d0| d1 | d2 | d3
```

(a) CFG

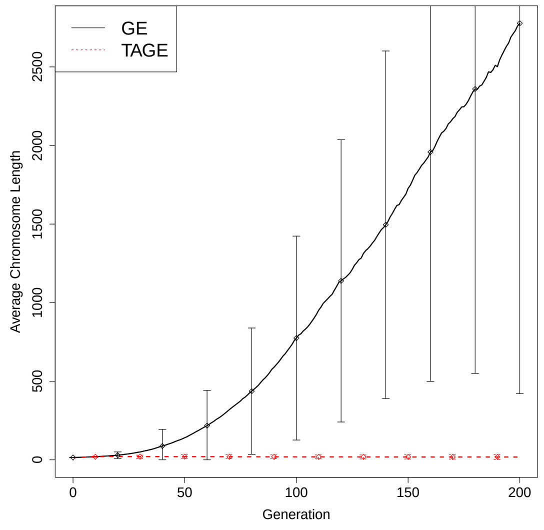


(b) Initial trees

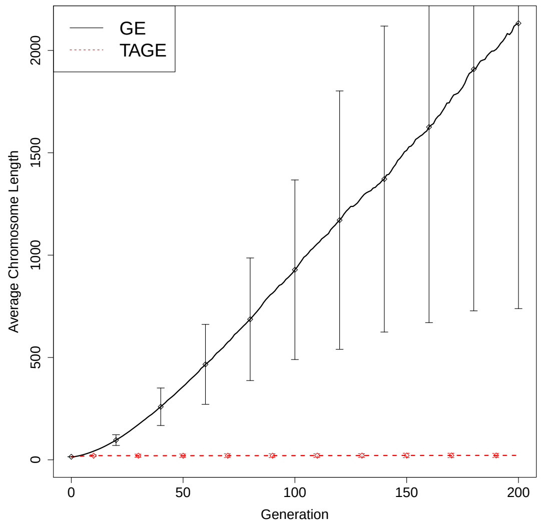


(c) Auxiliary trees

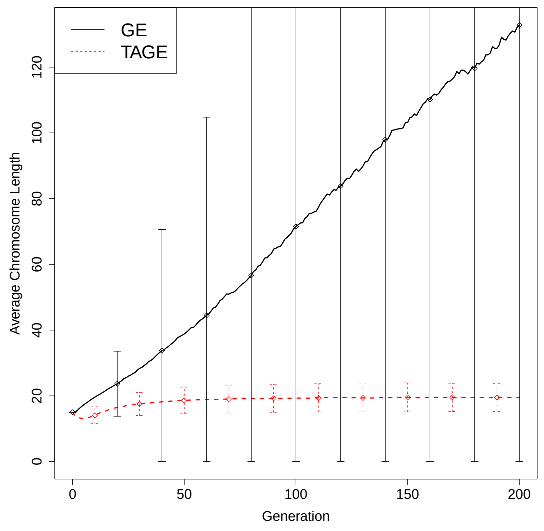
Figure B.4: CFG and TAG for the six multiplexer problem. The number in parenthesis indicates the amount of different variations of terminal productions that can be attached at that node.



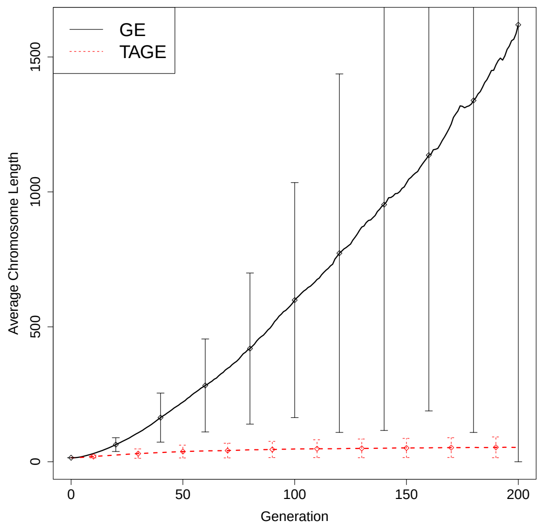
(a) Even five parity



(b) Santa Fe ant trail

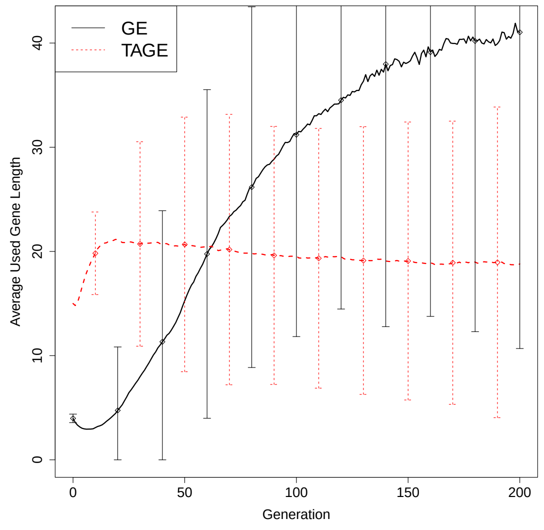


(c) Symbolic regression

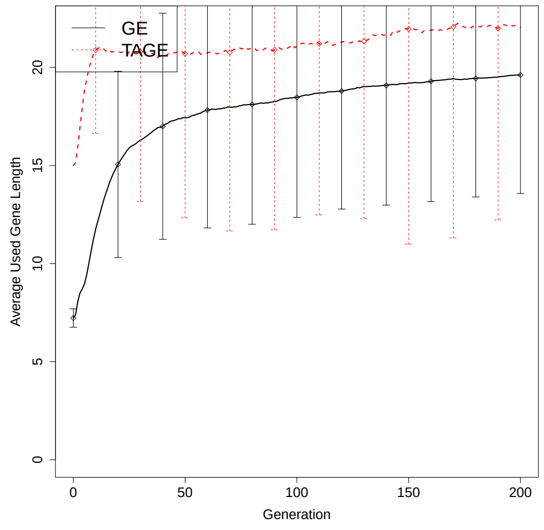


(d) Six multiplexer

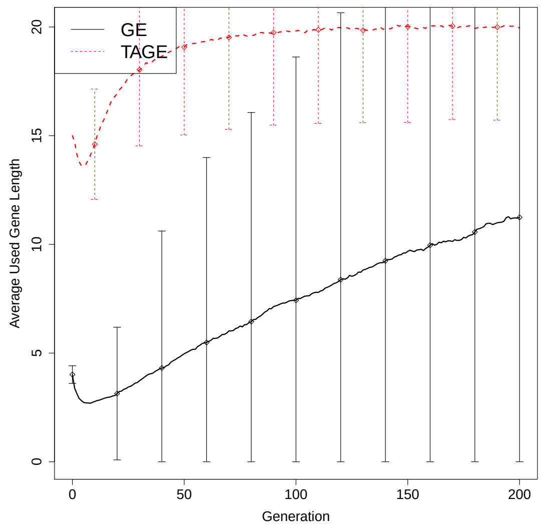
Figure B.5: Average length of chromosome plots across 500 runs at each generation with error bars of one standard deviation. Section 6.1.4.



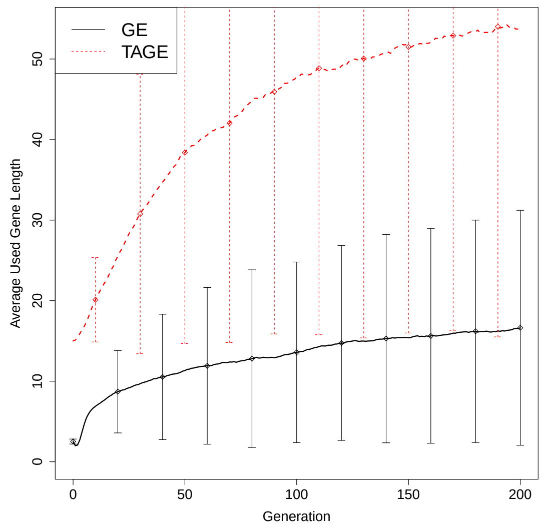
(a) Even five parity



(b) Santa Fe ant trail

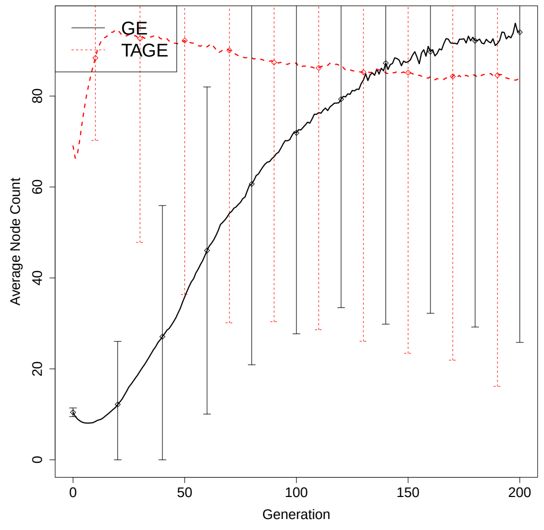


(c) Symbolic regression

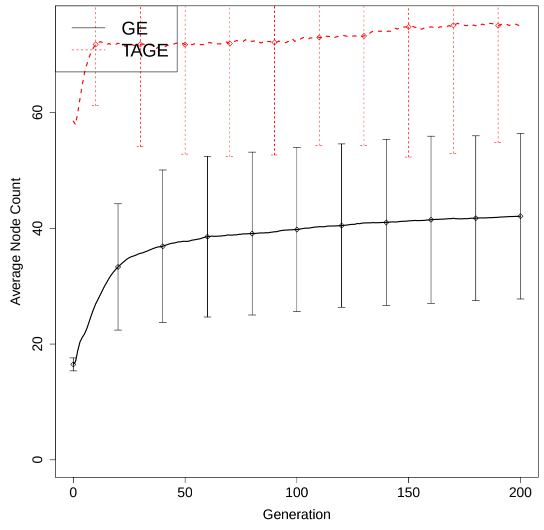


(d) Six multiplexer

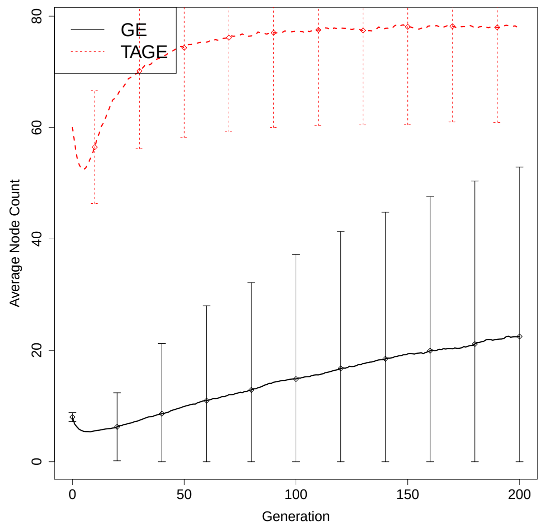
Figure B.6: Average length of effective region plots across 500 runs at each generation with error bars of one standard deviation. Section 6.1.4.



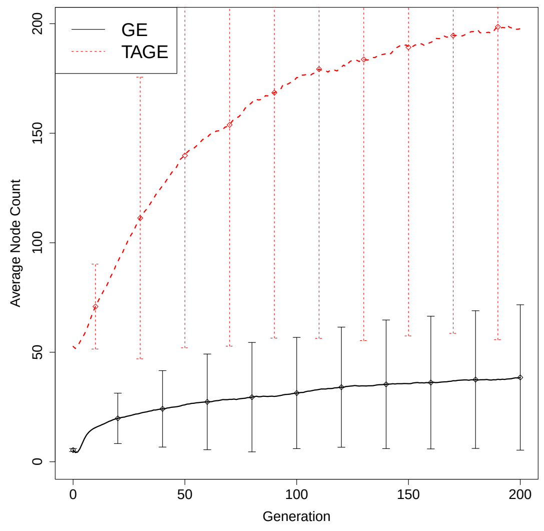
(a) Even five parity



(b) Santa Fe ant trail

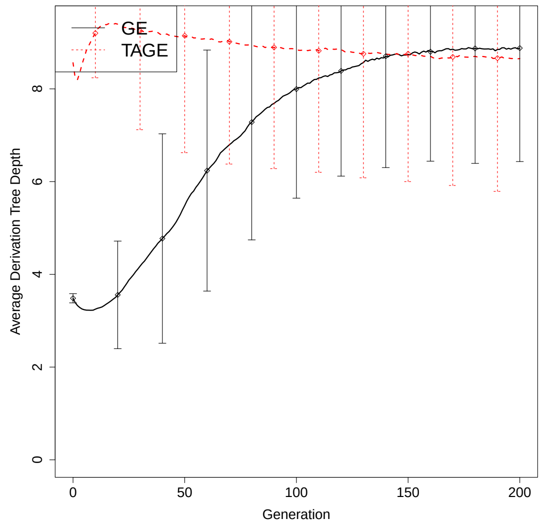


(c) Symbolic regression

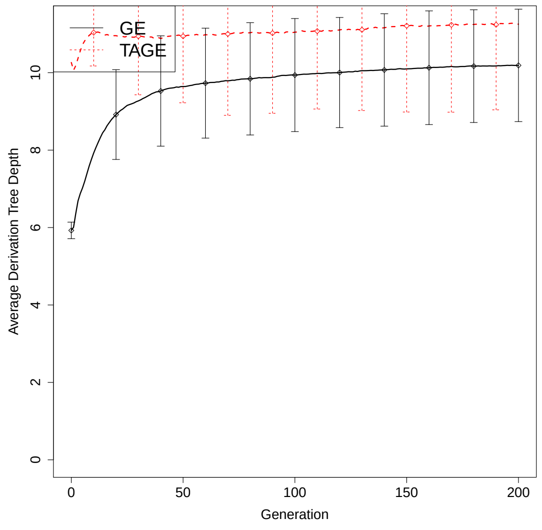


(d) Six multiplexer

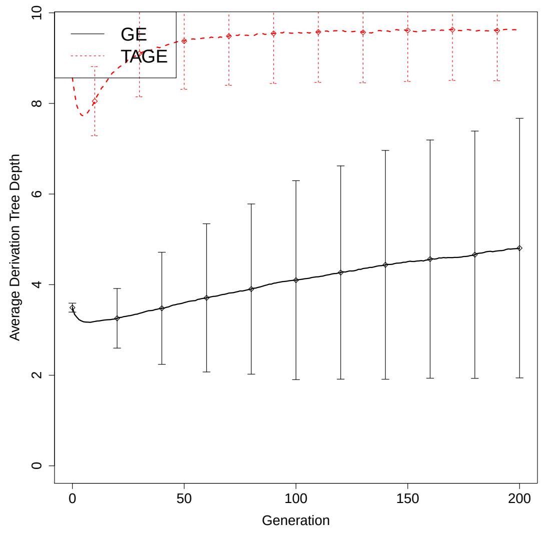
Figure B.7: Average derivation/derived tree node count plots across 500 runs at each generation with error bars of one standard deviation. Section 6.1.4.



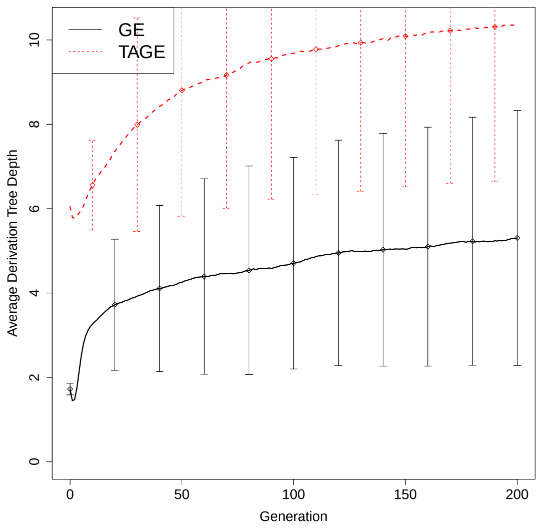
(a) Even five parity



(b) Santa Fe ant trail

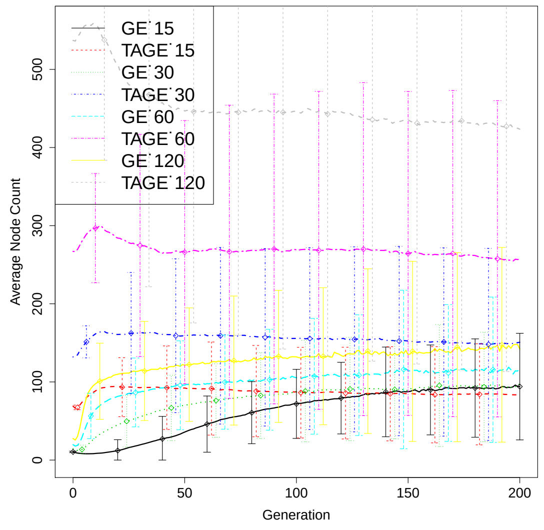


(c) Symbolic regression

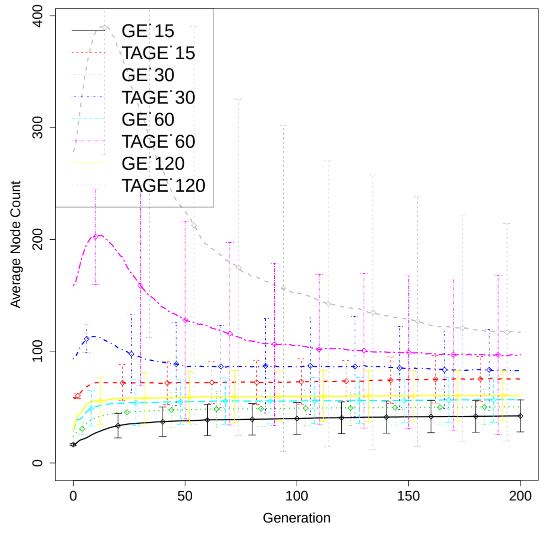


(d) Six multiplexer

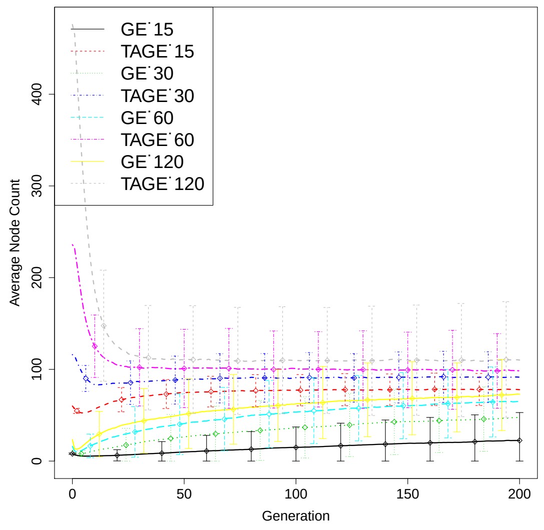
Figure B.8: Average derivation/derived tree depth plots across 500 runs at each generation with error bars of one standard deviation. Section 6.1.4.



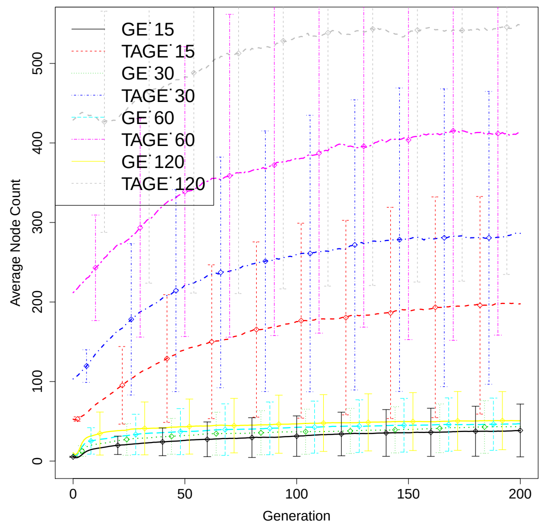
(a) Even five parity



(b) Santa Fe ant trail

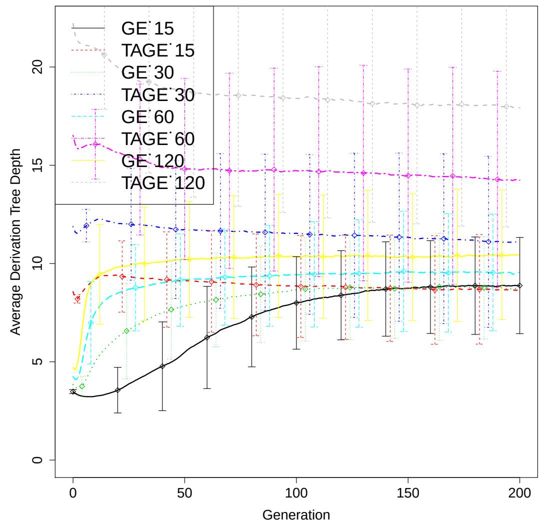


(c) Symbolic regression

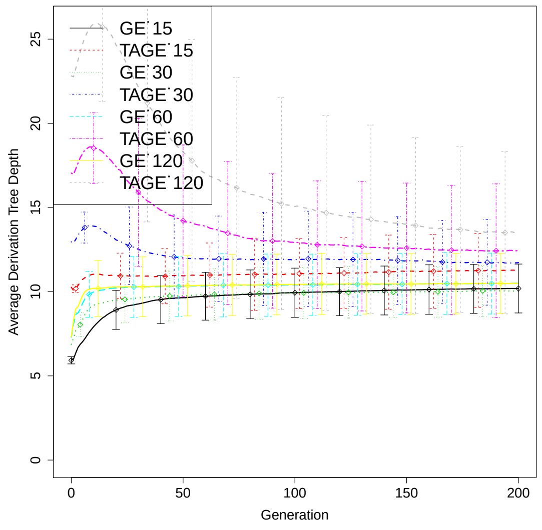


(d) Six multiplexer

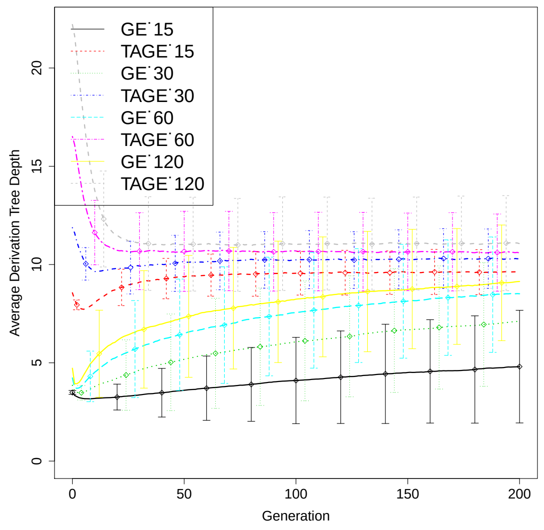
Figure B.9: Average derivation/derived tree node count plots across 500 runs at each generation with error bars of one standard deviation for varied initial chromosome lengths. Section 6.2.1.



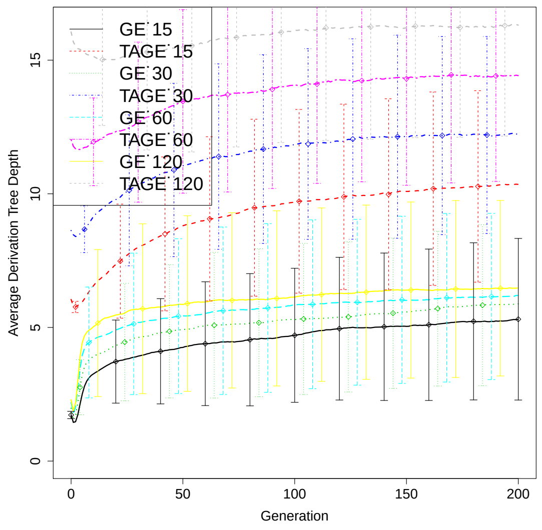
(a) Even five parity



(b) Santa Fe ant trail

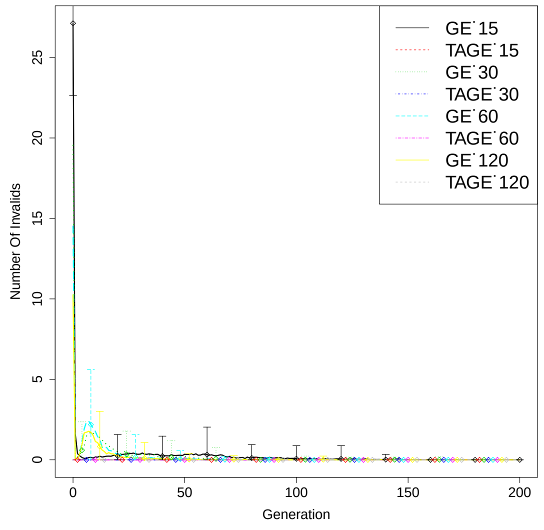


(c) Symbolic regression

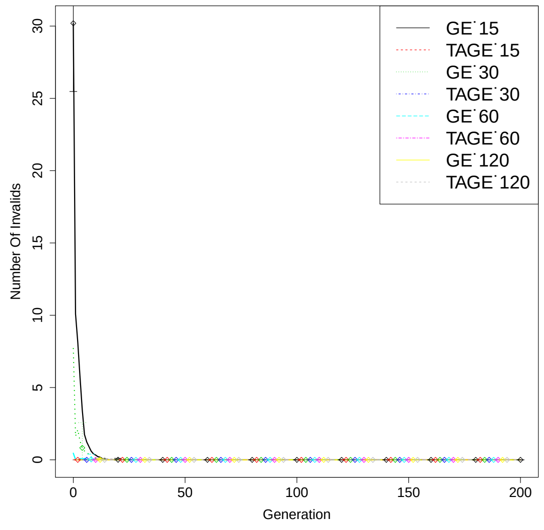


(d) Six multiplexer

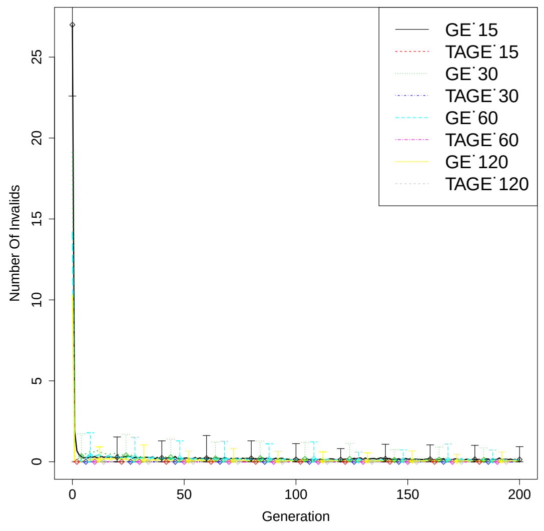
Figure B.10: Average derivation/derived tree depth plots across 500 runs at each generation with error bars of one standard deviation for varied initial chromosome lengths. Section 6.2.1.



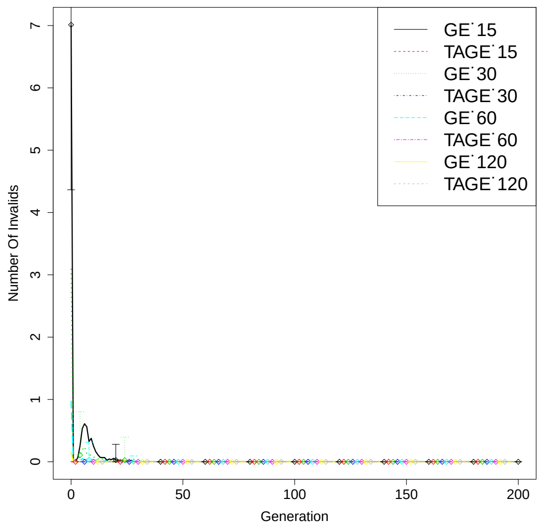
(a) Even five parity



(b) Santa Fe ant trail

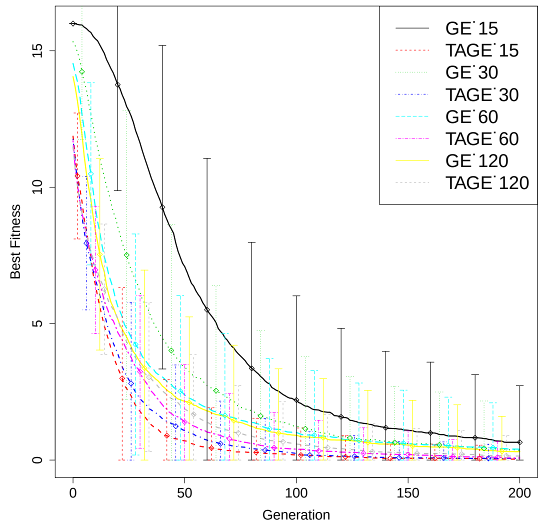


(c) Symbolic regression

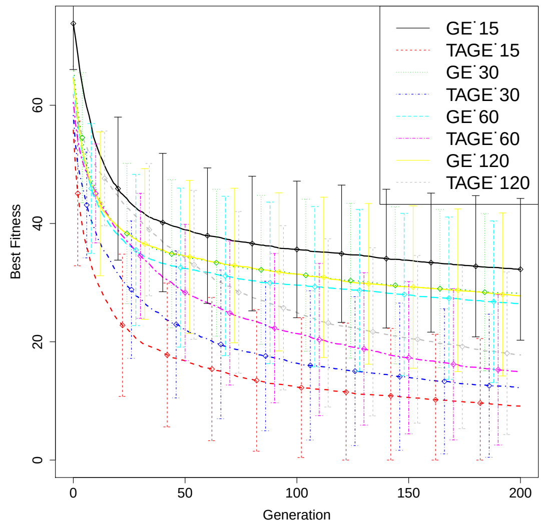


(d) Six multiplexer

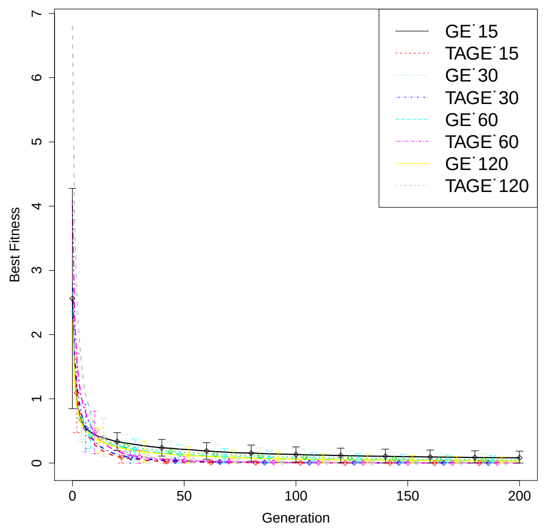
Figure B.11: Average invalid individual count plots across 500 runs at each generation with error bars of one standard deviation for varied initial chromosome lengths. Section 6.2.1.



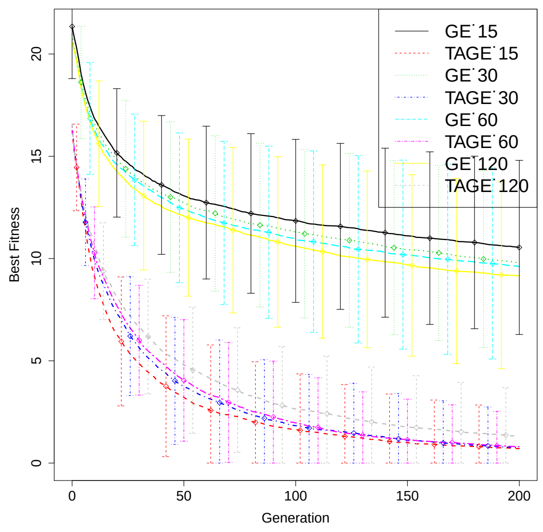
(a) Even five parity



(b) Santa Fe ant trail

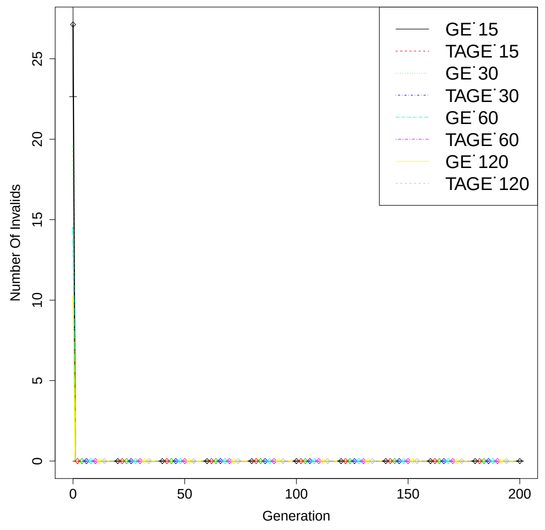


(c) Symbolic regression

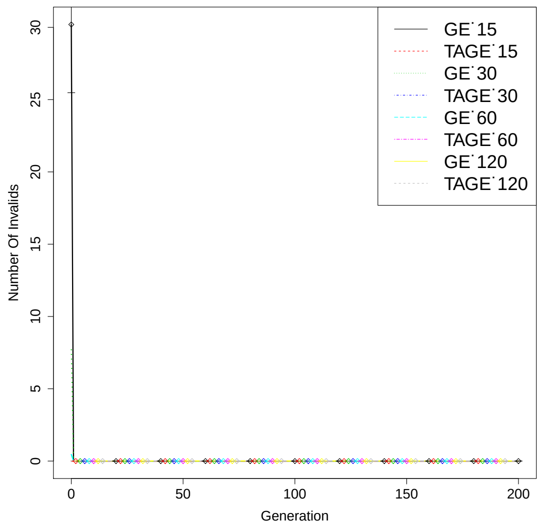


(d) Six multiplexer

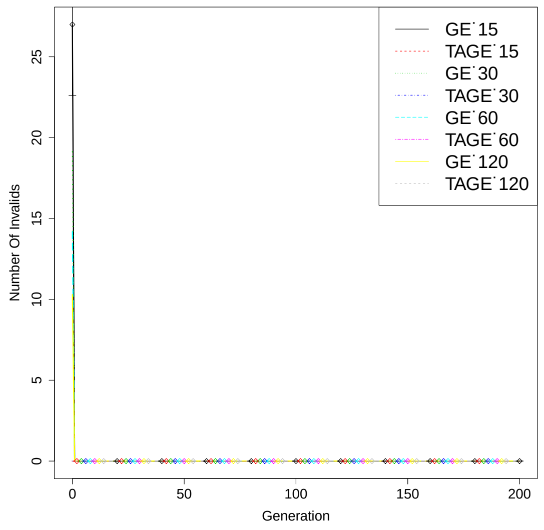
Figure B.12: Mean best fitness plots across 500 runs at each generation with error bars of one standard deviation for varied initial chromosome lengths with population validation enabled. Section 6.2.2.



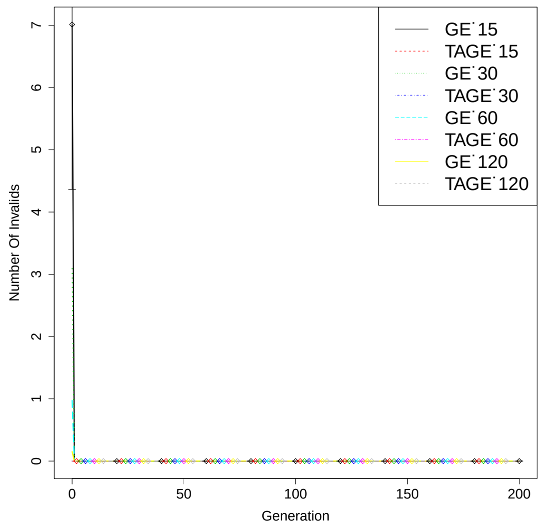
(a) Even five parity



(b) Santa Fe ant trail

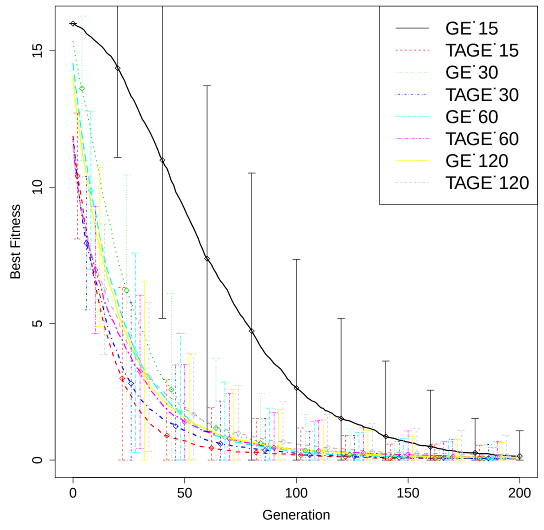


(c) Symbolic regression

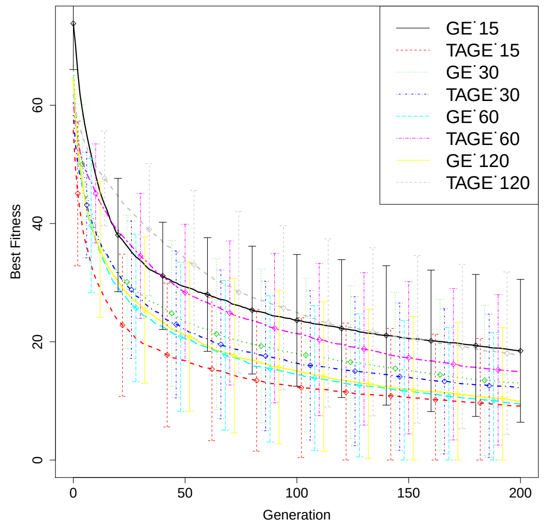


(d) Six multiplexer

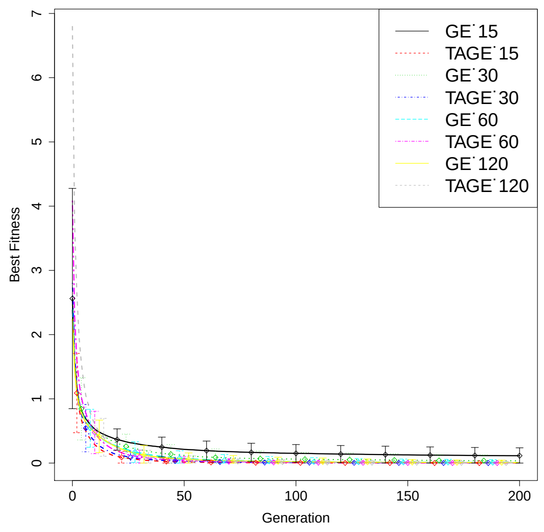
Figure B.13: Average invalid count plots across 500 runs at each generation with error bars of one standard deviation for varied initial chromosome lengths with population validation enabled. Section 6.2.2.



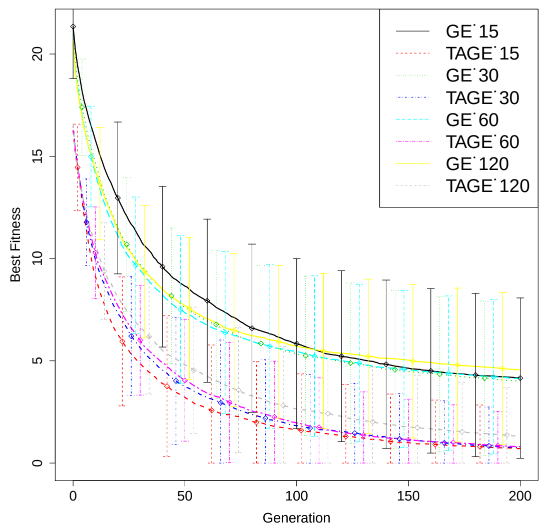
(a) Even five parity



(b) Santa Fe ant trail

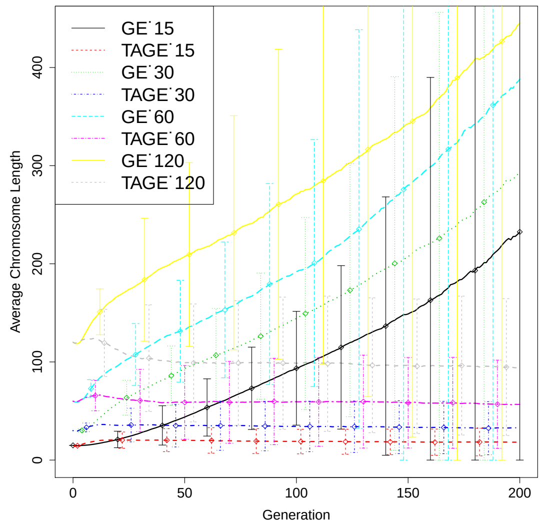


(c) Symbolic regression

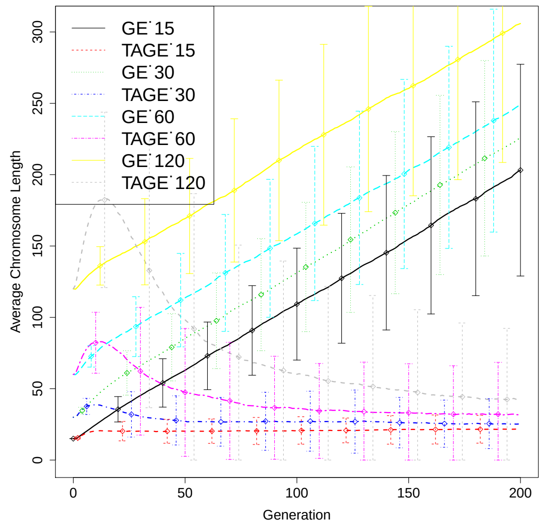


(d) Six multiplexer

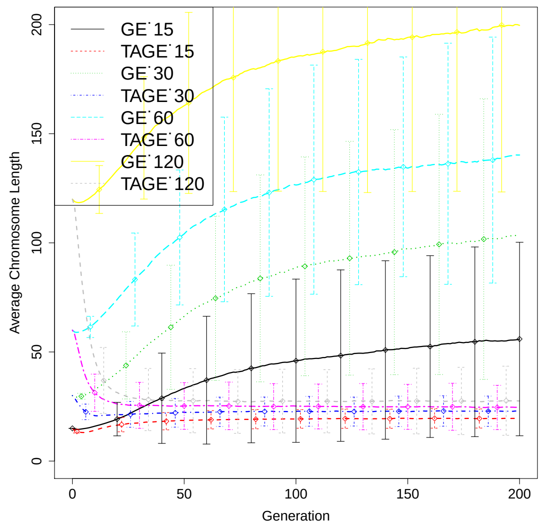
Figure B.14: Mean best fitness plots across 500 runs at each generation with error bars of one standard deviation for varied initial chromosome lengths with encoding region only crossover points. Section 6.2.3.



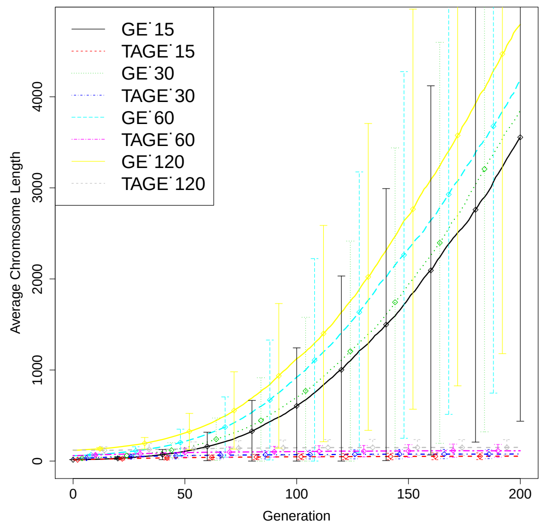
(a) Even five parity



(b) Santa Fe ant trail



(c) Symbolic regression



(d) Six multiplexer

Figure B.15: Average chromosome length plots across 500 runs at each generation with error bars of one standard deviation for varied initial chromosome lengths with encoding region only crossover points. Section 6.2.3.

Table B.1: Mann-Whitney U test results for best fitness values at generation 200. Shaded cells represent p-values < 0.05. Each cell is labelled with the p-value. Referred to from Section 6.1.3.

		GE
TAGE	EFP	1.0279e-11
	SF	3.7280e-133
	SR	1.1874e-96
	6 Mux	8.7847e-160

Table B.2: Mann-Whitney U test results for best fitness values at generation 200 for varied initial chromosome lengths. Shaded cells represent p-values < 0.05. Each cell is labelled with the p-value. Referred to from Section 6.2.1.

			GE			
			15	30	60	120
TAGE	EFP	15	1.0279e-11	4.0951e-08	9.3004e-10	5.3250e-06
		30	2.7032e-09	6.3823e-06	1.9349e-07	4.9839e-04
		60	9.5997e-07	8.2375e-04	4.2581e-05	2.5998e-02
		120	2.4303e-04	4.3253e-02	5.1033e-03	4.0323e-01
	SF	15	3.7280e-133	1.2401e-98	1.1236e-86	9.6392e-83
		30	5.0102e-119	5.5992e-81	8.3577e-67	7.5228e-64
		60	3.5211e-99	2.8648e-61	3.1251e-48	6.2772e-46
		120	2.2501e-75	4.1827e-42	9.2299e-31	2.8034e-29
	SR	15	1.1874e-96	2.8466e-32	2.1073e-21	2.0187e-19
		30	2.1167e-94	2.8840e-29	1.0251e-18	1.0962e-16
		60	1.5335e-91	3.7354e-26	1.6223e-15	1.4960e-13
		120	2.1345e-89	7.9815e-24	2.4474e-13	2.7058e-11
	6 Mux	15	8.7847e-160	1.0971e-150	4.7334e-150	1.2696e-143
		30	6.2415e-160	5.5224e-151	2.2581e-150	1.1550e-143
		60	3.0512e-158	4.6491e-149	1.1613e-148	1.5034e-141
		120	5.3027e-147	5.2853e-137	3.2739e-135	3.2421e-127

Table B.3: Mann-Whitney U test results for best fitness values at generation 200 for varied initial chromosome lengths with population validation enabled. Shaded cells represent p-values < 0.05. Each cell is labelled with the p-value. Referred to from Section 6.2.2.

			GE			
			15	30	60	120
TAGE	EFP	15	1.7450e-10	2.4615e-05	2.2906e-07	4.2279e-07
		30	3.9548e-08	1.8661e-03	3.0477e-05	5.4056e-05
		60	1.0563e-05	6.7589e-02	3.0032e-03	4.8597e-03
		120	1.7134e-03	6.6284e-01	1.0979e-01	1.5544e-01
	SF	15	5.6099e-120	2.8297e-91	4.8807e-80	3.1486e-88
		30	1.2757e-103	2.9340e-73	1.2432e-59	4.4471e-69
		60	6.4545e-84	1.6850e-54	1.3657e-41	1.6545e-50
		120	4.5229e-62	4.1910e-36	3.8938e-25	6.5413e-33
	SR	15	7.6728e-17	2.1266e-08	7.6393e-08	5.1703e-08
		30	1.2912e-14	1.8651e-06	8.1282e-06	5.9041e-06
		60	7.5892e-12	2.3571e-04	9.2483e-04	8.6917e-04
		120	2.1648e-10	2.7040e-03	1.1506e-02	1.1999e-02
	6 Mux	15	2.2470e-162	1.3414e-151	1.9281e-146	1.5832e-145
		30	5.7901e-163	5.3060e-152	2.4123e-146	1.4987e-145
		60	8.4480e-162	2.5518e-150	2.3144e-144	1.3135e-143
		120	1.1740e-149	3.4948e-137	2.6406e-130	3.1826e-128

Table B.4: Mann-Whitney U test results for best fitness values at generation 200 for varied initial chromosome lengths with encoding region restricted crossover. Shaded cells represent p-values < 0.05. Each cell is labelled with the p-value. Referred to from Section 6.2.3.

			GE			
			15	30	60	120
TAGE	EFP	15	1.7197e-02	7.3838e-01	7.4445e-01	2.9085e-02
		30	2.8019e-01	2.8602e-01	2.8191e-01	3.9305e-01
		60	7.5450e-01	1.6260e-02	1.5700e-02	5.7773e-01
		120	8.3366e-02	2.2824e-04	2.1250e-04	4.6203e-02
	SF	15	8.9006e-44	6.1081e-11	4.5025e-02	7.3353e-02
		30	1.6138e-22	9.8481e-03	4.9001e-02	4.0797e-02
		60	1.7980e-09	2.3898e-01	7.3956e-09	1.5881e-08
		120	2.1289e-02	1.4518e-05	5.1002e-18	1.4973e-17
	SR	15	8.2371e-50	1.3060e-08	2.4576e-08	1.0252e-07
		30	1.8637e-39	3.1884e-03	3.1920e-02	4.2950e-02
		60	2.5728e-31	6.1997e-01	4.5060e-01	3.8873e-01
		120	7.2128e-27	3.6283e-01	1.0354e-02	7.8624e-03
	6 Mux	15	3.3274e-60	2.5697e-60	2.9132e-65	5.7979e-77
		30	1.1524e-57	7.9648e-58	1.5372e-62	1.0297e-74
		60	2.9941e-52	1.2534e-52	3.1133e-57	4.8242e-70
		120	3.6046e-35	5.5157e-35	8.9333e-39	5.1402e-50

Table B.5: Mann-Whitney U test results for best fitness values at generation 200 with varied initial chromosome size, population validation enabled and encoding region restricted crossover. Shaded cells represent p-values < 0.05. Each cell is labelled with the p-value. Referred to from Section 6.2.4.

			GE			
			15	30	60	120
TAGE	EFP	10	9.3212e-04	2.5170e-03	2.9620e-08	1.1746e-09
		30	3.5210e-02	7.4152e-02	4.8272e-06	2.5082e-07
		60	4.4890e-01	6.7431e-01	6.8167e-04	5.7143e-05
		120	4.9992e-01	3.1101e-01	4.3249e-02	7.5110e-03
	SF	15	3.1725e-25	1.2617e-04	6.5877e-03	1.4781e-02
		30	2.5255e-10	8.7082e-01	2.0462e-01	1.2321e-01
		60	8.3863e-03	8.6187e-05	4.1119e-07	1.3373e-07
		120	4.1428e-01	3.0921e-12	1.2900e-15	1.6491e-16
	SR	15	1.5233e-18	5.5861e-05	8.8809e-06	2.6288e-11
		30	2.3612e-12	5.8969e-02	4.1191e-02	5.9232e-05
		60	1.3404e-07	7.9905e-01	7.6050e-01	9.2522e-02
		120	1.8633e-05	1.3681e-01	9.8822e-02	8.7912e-01
	6 Mux	15	2.1726e-70	1.2148e-77	1.3679e-73	1.1408e-76
		30	3.0585e-68	2.5846e-75	3.6098e-71	1.8702e-74
		60	4.4024e-63	1.0616e-70	1.9015e-66	8.2368e-70
		120	1.8496e-45	1.6167e-50	8.6007e-47	1.2669e-49

Appendix C

Chapter 8

Algorithm C.1 A recursive algorithm for calculating a GE chromosome retrospectively from an existing derivation tree or TAG derived tree.

```
Require: $rootNode$ {Derivation tree root, or sub-root node}
Require: $chromosome$ {Reference to the chromosome}
Require: $grammar$ {A sufficient CFG}
  {Recursive base case}
  $symbol \leftarrow rootNode$.getSymbol
  if $symbol$.getType = terminal then
    return
  {Only generate a codon if there is a choice to be made}
  $rule \leftarrow grammar$.getRule($symbol$)
  if $rule$.size $> 1$ then
    $productionIndex \leftarrow -1$
    $match \leftarrow$ false
    while $productionIndex < rule$.size and not $match$ do
      $productionIndex \leftarrow productionIndex + 1$
      {Check each symbol in the production against child nodes}
      $p \leftarrow rule$.get($productionIndex$)
      if $p$.compareEachSymbolToNodes($rootNode$.getChildren) then
        $match \leftarrow$ true
    {Generate codon and add to the chromosome}
    $codon \leftarrow$genRandomCodon($productionIndex$, $rule$.size)
    $chromosome$.add($codon$)
  {Recurse in a depth-first manner}
  for all $n_i$ in $rootNode$.getChildren do
    recurse($n_i$, $chromosome$, $grammar$)
```

Table C.1: Mann-Whitney U test results for best fitness values at generation 200. Shaded cells represent p-values < 0.05. Cells are labelled with the p-values for Table 8.3.

GE				Method	TAGE			
1.00e+00	6.96e-03	3.98e-02	1.76e-02	R015	1.0e+00	4.96e-04	1.00e+00	1.00e+00
6.96e-03	1.00e+00	1.00e+00	1.00e+00	R120	4.96e-04	1.00e+00	1.56e+02	7.93e-03
3.98e-02	1.00e+00	1.00e+00	1.00e+00	RNTS	1.00e+00	1.56e+02	1.00e+00	1.00e+00
1.76e-02	1.00e+00	1.00e+00	1.00e+00	RRT	1.00e+00	7.93e-03	1.00e+00	1.00e+00

(a) Even five parity

GE				Method	TAGE			
1.00e+00	3.01e-14	3.11e-05	9.64e-14	R015	1.00e+00	4.40e-25	2.76e-03	1.00e+00
3.01e-14	1.00e+00	5.83e-04	1.00e+00	R120	4.40e-25	1.00e+00	4.36e-13	1.05e-21
3.11e-05	5.83e-04	1.00e+00	5.30e-04	RNTS	2.76e-03	4.36e-13	1.00e+00	5.28e-02
9.64e-14	1.00e+00	5.30e-04	1.00e+00	RRT	1.00e+00	1.05e-21	5.28e-02	1.00e+00

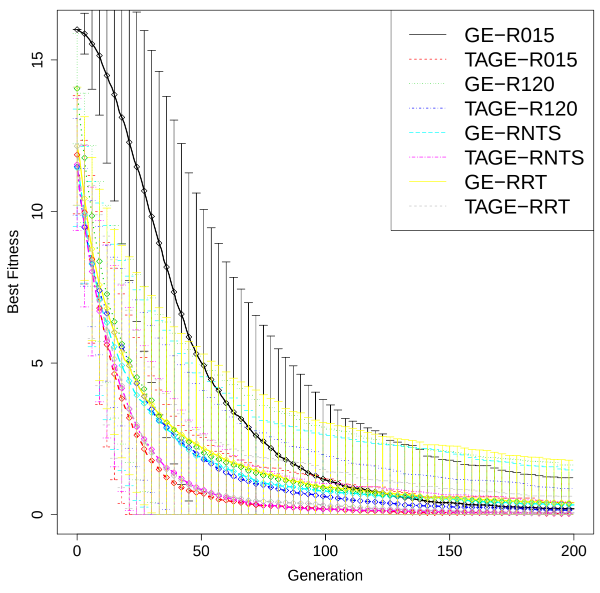
(b) Santa Fe ant trail

GE				Method	TAGE			
1.00e+00	1.33e-04	1.00e+00	1.78e-02	R015	1.00e+00	5.02e-14	1.02e-01	1.00e+00
1.33e-04	1.00e+00	3.04e-03	5.40e-01	R120	5.02e-14	1.00e+00	4.35e-07	2.92e-12
1.00e+00	3.04e-03	1.00e+00	4.68e-01	RNTS	1.02e-01	4.35e-07	1.00e+00	3.49e-01
1.78e-02	5.40e-01	4.68e-01	1.00e+00	RRT	1.00e+00	2.92e-12	3.49e-01	1.00e+00

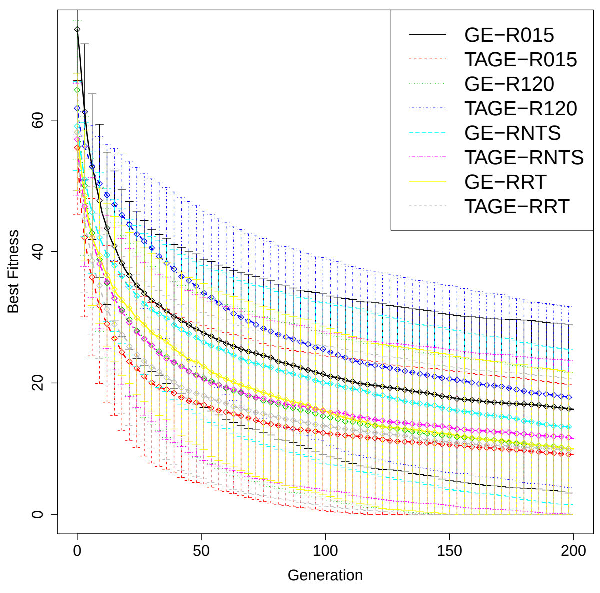
(c) Symbolic regression

GE				Method	TAGE			
1.00e+00	1.00e+00	1.00e+00	3.85e-02	R015	1.00e+00	8.90e-09	1.00e+00	4.47e-01
1.00e+00	1.00e+00	1.00e+00	1.09e-01	R120	8.90e-09	1.00e+00	2.00e-08	4.02e-14
1.00e+00	1.00e+00	1.00e+00	4.96e-02	RNTS	1.00e+00	2.00e-08	1.00e+00	1.96e-01
3.85e-02	1.09e-01	4.96e-02	1.00e+00	RRT	4.47e-01	4.02e-14	1.96e-01	1.00e+00

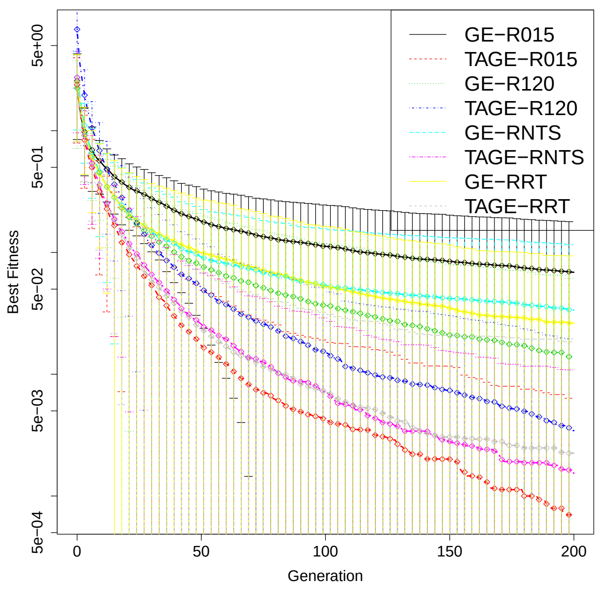
(d) Six multiplexer



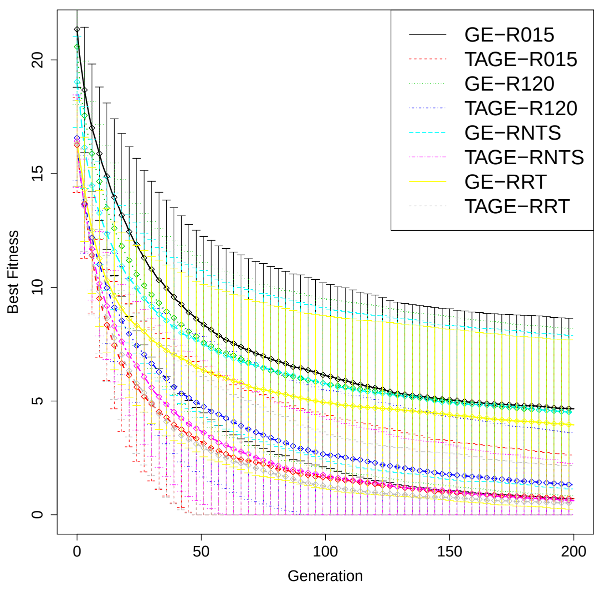
(a) Even five parity



(b) Santa Fe ant trail

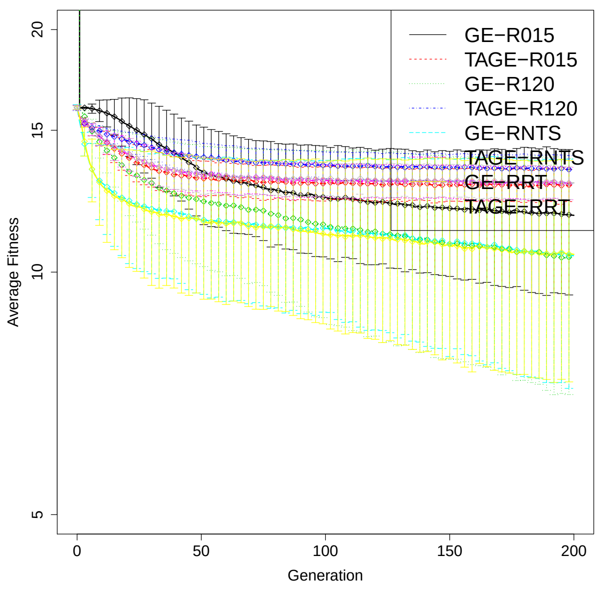


(c) Symbolic regression

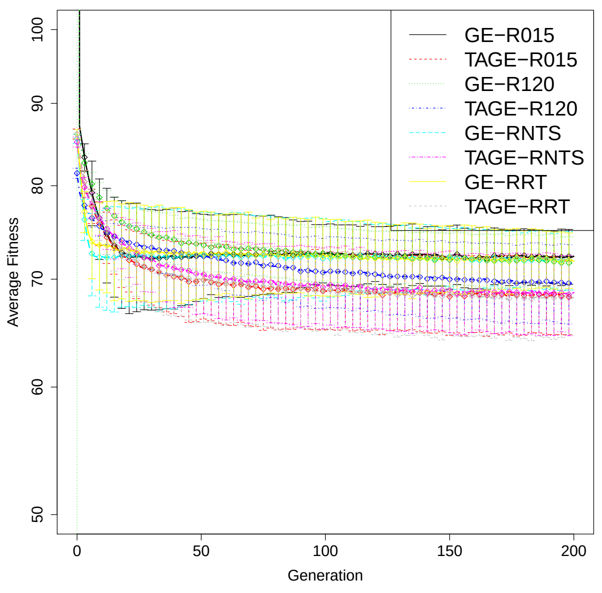


(d) Six multiplexer

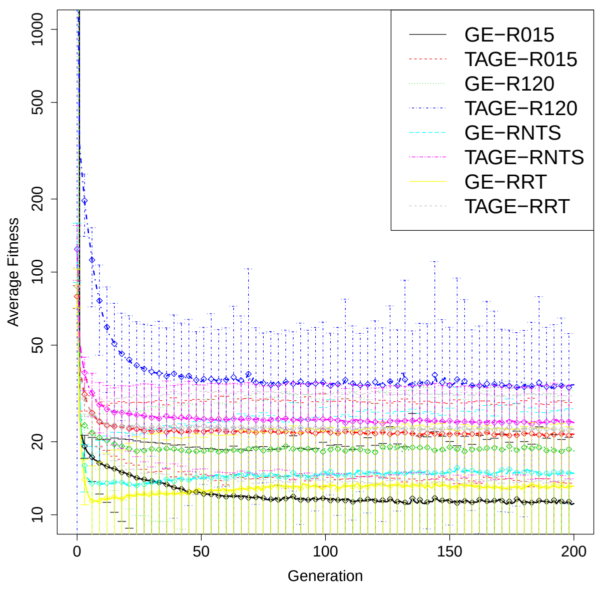
Figure C.1: Mean best fitness plots for the all problems with error bars of one standard deviation.



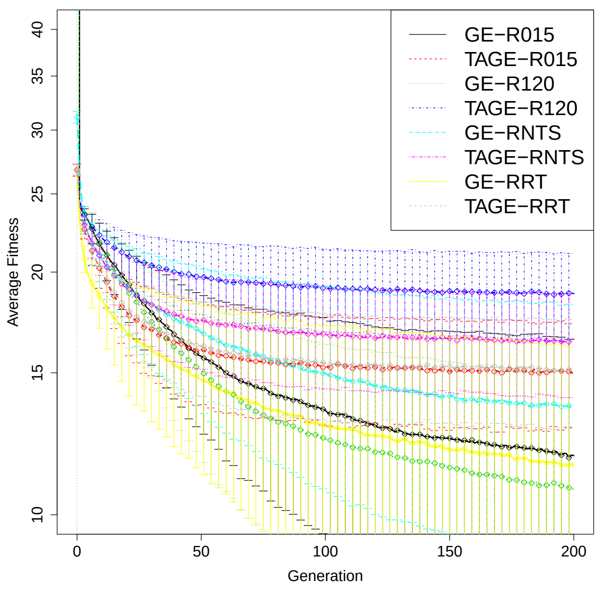
(a) Even five parity



(b) Santa Fe ant trail



(c) Symbolic regression



(d) Six multiplexer

Figure C.2: Mean population fitness plots for the all problems with error bars of one standard deviation.

Table C.2: Mann-Whitney U test results for best fitness values at generation 200. Shaded cells represent p-values < 0.05. Each cell is labelled with the p-value. Referred to from Table 8.4.

		GE
TAGE	EFP	1.7091e-07
	SF	0.1535
	SR	1.3339e-12
	6 Mux	1.4397e-65

Table C.3: Mann-Whitney U test results for best fitness values at generation 200. Shaded cells represent p-values < 0.05. Each cell is labelled with the p-value. Referred to from Table 8.6.

		Orig. Adjunction Biases
Mod.	EFP	3.4627e-01
	SF	1.6709e-02
	SR	7.7094e-02
	6 Mux	6.8533e-01

Table C.4: Mann-Whitney U test results for best fitness values at generation 200. Shaded cells represent p-values < 0.05. Each cell is labelled with the p-value. Referred to from Table 8.6.

		GE
TAGE	EFP	3.9164e-04
	SF	5.4002e-03
	SR	7.0494e-24
	6 Mux	1.8243e-42

Table C.5: Mann-Whitney U test results for best fitness values at generation 200. Shaded cells represent p-values < 0.05. Each cell is labelled with the p-value. Referred to from Table 8.7.

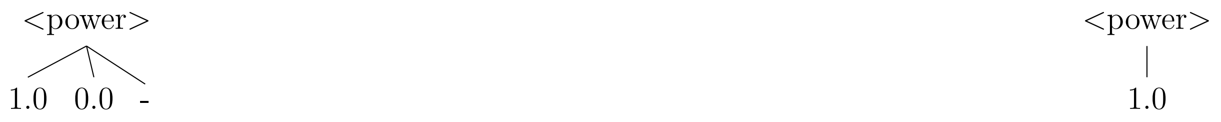
		TAGE
PTAGE	EFP	9.9957e-01
	SF	4.2404e-06
	SR	9.9991e-01
	6 Mux	1.5381e-39

Appendix D

Chapter 9

```
<power>  ::= 1.0 0.0 - | 1.0
```

(a) CFG

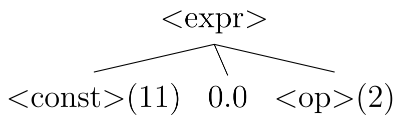


(b) Initial trees

Figure D.1: CFG and TAG for directly mapping to power factors of 1.0 and -1.0 for the inverted-pendulum problem. The grammar is not recursive, there are not TAG auxiliary trees.

```
<expr>  ::= <const> 0.0 <op>
<op>    ::= + | -
<const> ::= 0.0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5
          | 0.6 | 0.7 | 0.8 | 0.9 | 1.0
```

(a) CFG

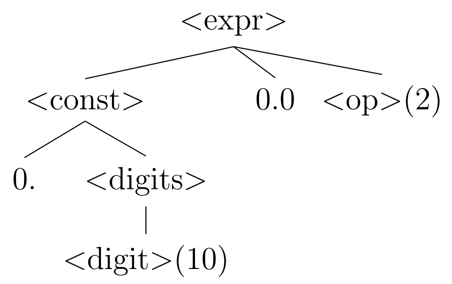


(b) Initial trees

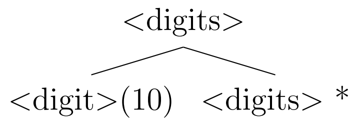
Figure D.2: CFG and TAG for mapping to a discrete set of power factors in the interval $[-1, 1]$ for the inverted-pendulum problem. The grammar is not recursive, there are not TAG auxiliary trees.

```
<expr>  ::= <const> 0.0 <op>
<op>    ::= + | -
<const> ::= 0.<digits>
<digits> ::= <digit><digits> | <digit>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

(a) CFG



(b) Initial trees

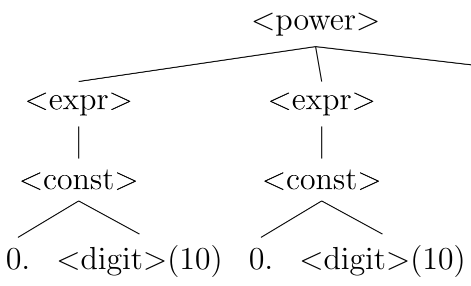


(c) Auxiliary trees

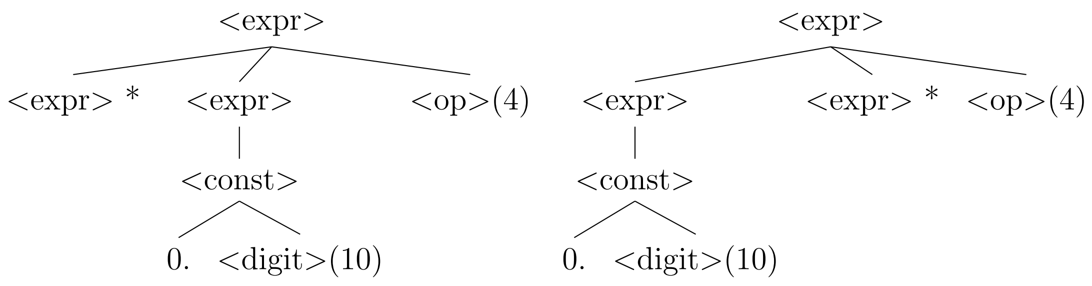
Figure D.3: A recursive CFG and TAG for mapping to a continuous set of power factors in the interval $(-1, 1)$ for the inverted-pendulum problem. The number in parenthesis indicates the amount of different variations of terminal productions that can be attached at that node.

```
<power> ::= <expr> <expr> <op>
<expr>  ::= <expr> <expr> <op> | <const>
<op>    ::= + | - | * | "/"
<const> ::= 0.<digit>
<digit> ::= 0 | 1 | 2 | 3 | 4
          | 5 | 6 | 7 | 8 | 9
```

(a) CFG



(b) Initial trees



(c) Auxiliary trees

Figure D.4: A recursive CFG and TAG for mapping to expressions for generating power factors for the inverted-pendulum problem. Factors are clamped between $[-1, 1]$. The number in parenthesis indicates the amount of different variations of terminal productions that can be attached at that node.

Bibliography

[1] H. Abbass, N. X. Hoai, and R. I. (Bob) McKay. AntTAG: A new method to compose computer programs using colonies of ants. In *Proceedings, 2002 World Congress on Computational Intelligence*, volume 2, pages 1654–1666. IEEE Press, 2002. doi: 10.1109/CEC.2002.1004490.

[2] Raja Muhammad Atif Azad. *A Position Independent Representation for Evolutionary Automatic Programming Algorithms - The Chorus System.* PhD thesis, University of Limerick, Ireland, December 2003.

[3] J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, and M. Woodger. Revised report on the algorithm language algol 60. *Commun. ACM*, 6(1):1–17, January 1963. ISSN 0001-0782. doi: 10.1145/366193.366201.

[4] Wolfgang Banzhaf. On the dynamics of an artificial regulatory network. In Wolfgang Banzhaf, Jens Ziegler, Thomas Christaller, Peter Dittrich, and JanT. Kim, editors, *Advances in Artificial Life*, volume 2801 of *Lecture Notes in Computer Science*, pages 217–227. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-20057-4.

[5] Wolfgang Banzhaf. Artificial regulatory networks and genetic programming. In *Genetic Programming Theory and Practice*, chapter 4, pages 43–62. Kluwer, 2003.

[6] Wolfgang Banzhaf. On evolutionary design, embodiment, and artificial regulatory networks. In *Embodied Artificial Intelligence*, pages 284–292. SpringerVerlag, 2004.

[7] Wolfgang Banzhaf and P. Dwight Kuo. Network motifs in natural and artificial transcriptional regulatory networks. *Journal of Biological Physics and Chemistry*, 4: 85–92, 2004.

[8] Wolfgang Banzhaf, Frank D. Francone, Robert E. Keller, and Peter Nordin. *Genetic programming: an introduction: on the automatic evolution of computer programs and its applications.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998. ISBN 1-55860-510-X.

[9] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, & Cybernetics*, pages 834–846, Sep-Oct 1983.

[10] Lewin Benjamin. Genes vii, 2000.

[11] Peter J. Bentley. Fractal proteins. In *Genetic Programming and Evolvable Machines*, pages 5–71. Springer Verlag, 2004.

[12] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies a comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002. ISSN 1567-7818. doi: 10.1023/A:1015059928466.

[13] J. Bongard. Evolving modular genetic regulatory networks. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, volume 2, pages 1872–1877, 2002. doi: 10.1109/CEC.2002.1004528.

[14] A. Brabazon, M. O'Neill, and D. G. Maringer, editors. *Natural Computing in Computational Finance (Volume 3)*, volume 293 of *Studies in Computational Intelligence*. Springer, 2010.

[15] Anthony Brabazon and Michael O'Neill. Anticipating bankruptcy reorganisation from raw financial data using grammatical evolution. In *Applications of Evolutionary Computing*, pages 368–377. Springer, 2003.

[16] Anthony Brabazon and Michael O'Neill. Diagnosing corporate stability using grammatical evolution. *International Journal of Applied Mathematics and Computer Science*, 14(3):363–374, 2004.

[17] Anthony Brabazon and Michael O'Neill. Bond-issuer credit rating with grammatical evolution. In Guenther R. Raidl, Stefano Cagnoni, Jurgen Branke, David W. Corne, Rolf Drechsler, Yaochu Jin, Colin R. Johnson, Penousal Machado, Elena Marchiori, Franz Rothlauf, George D. Smith, and Giovanni Squillero, editors, *Applications of Evolutionary Computing, EvoWorkshops2004: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, EvoSTOC*, volume 3005 of *LNCS*, pages 270–279, Coimbra, Portugal, 5-7 April 2004. Springer Verlag. ISBN 3-540-21378-3. doi: 10.1007/978-3-540-24653-4_28.

[18] Anthony Brabazon and Michael O'Neill. Credit rating with pi grammatical evolution. In R. Tadeusiewicz, A. Ligeza, and M. Szymkat, editors, *Proceedings of Computer Methods and Systems Conference*, volume 1, pages 253–260, Krakow, Poland, 14-16 November 2005. Oprogramowanie Naukowo-Techniczne Tadeusiewicz. ISBN 83-916420-3-8.

[19] Anthony Brabazon and Michael O'Neill. *Biologically Inspired Algorithms for Financial Modelling*. Natural Computing Series. Springer, 2006. ISBN 3-540-26252-0. doi: 10.1007/3-540-31307-9.

[20] Anthony Brabazon and Michael O'Neill. Credit classification using grammatical evolution. *Informatica*, 30(3):325–335, 2006. ISSN 0350-5596.

[21] Anthony Brabazon, Michael O'Neill, Robin Matthews, and Conor Ryan. Grammatical evolution and corporate failure prediction. In *GECCO*, volume 2, pages 1011–1018, 2002.

[22] Anthony Brabazon, Katrina Meagher, Edward Carty, Michael O'Neill, and Peter Keenan. Grammar-mediated time-series prediction. *Journal of Intelligent Systems*, 14(2–3):123–143, August 2004. ISSN 0334-1860. doi: 10.1515/JISYS.2005.14.2-3.123.

[23] Tony Brabazon and Michael O'Neill. Trading foreign exchange markets using evolutionary automatic programming. In Alwyn M. Barry, editor, *GECCO 2002: Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference*, pages 133–136, New York, 8 July 2002. AAAI.

[24] Tony Brabazon, M. O'Neill, C. Ryan, and J. J. Collins. Uncovering technical trading rules using evolutionary automatic programming. In *Proceedings of 2001 AAANZ Conference (Accounting Association of Australia and NZ)*, Auckland, New Zealand, 1-3 July 2001.

[25] Robert Gregory Bradley, Anthony Brabazon, and Michael O'Neill. Evolving trading rule-based policies. In Cecilia Di Chio, Anthony Brabazon, Gianni A. Di Caro, Marc Ebner, Muddassar Farooq, Andreas Fink, Jorn Grahl, Gary Greenfield, Penousal Machado, Michael O'Neill, Ernesto Tarantino, and Neil Urquhart, editors, *EvoFIN*, volume 6025 of *LNCS*, pages 251–260, Istanbul, 7-9 April 2010. Springer. doi: 10.1007/978-3-642-12242-2_26.

[26] Robert Burbidge, Joanne H. Walker, and Myra S. Wilson. Grammatical evolution of a robot controller. In *Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems*, IROS'09, pages 357–362, Piscataway, NJ, USA, 2009. IEEE Press. ISBN 978-1-4244-3803-7.

[27] Jonathan Byrne. *Approaches to Evolutionary Architectural Design Exploration Using Grammatical Evolution*. PhD thesis, University College Dublin, 2012.

[28] Jonathan Byrne, Michael O'Neill, and Anthony Brabazon. Structural and nodal mutation in grammatical evolution. In Guenther Raidl, Franz Rothlauf, Giovanni Squillero, Rolf Drechsler, Thomas Stuetzle, Mauro Birattari, Clare Bates Congdon, Martin Middendorf, Christian Blum, Carlos Cotta, Peter Bosman, Joern Grahl, Joshua Knowles, David Corne, Hans-Georg Beyer, Ken Stanley, Julian F. Miller, Jano van Hemert, Tom Lenaerts, Marc Ebner, Jaume Bacardit, Michael O'Neill, Massimiliano Di Penta, Benjamin Doerr, Thomas Jansen, Riccardo Poli, and Enrique Alba, editors, *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1881–1882, Montreal, 8-12 July 2009. ACM. doi: 10.1145/1569901.1570215.

[29] Jonathan Byrne, James McDermott, Edgar Galvan-Lopez, and Michael O'Neill. Implementing an intuitive mutation operator for interactive evolutionary 3D design.

In *2010 IEEE World Congress on Computational Intelligence*, pages 2919–2925, Barcelona, Spain, 18-23 July 2010. IEEE Computational Intelligence Society, IEEE Press. doi: 10.1109/CEC.2010.5586485.

[30] Jonathan Byrne, James McDermott, Michael O'Neill, and Anthony Brabazon. An analysis of the behaviour of mutation in grammatical evolution. In Anna Isabel Esparcia-Alcazar, Aniko Ekart, Sara Silva, Stephen Dignum, and A. Sima Uyar, editors, *Proceedings of the 13th European Conference on Genetic Programming, EuroGP 2010*, volume 6021 of *LNCS*, pages 14–25, Istanbul, 7-9 April 2010. Springer. doi: 10.1007/978-3-642-12148-7_2.

[31] Jonathan Byrne, Michael Fenton, Erik Hemberg, James McDermott, Michael O'Neill, Elizabeth Shotton, and Ciaran Nally. Combining structural analysis and multi-objective criteria for evolutionary architectural design. In Cecilia Di Chio, Anthony Brabazon, Gianni Di Caro, Rolf Drechsler, Marc Ebner, Muddassar Farooq, Joern Grahl, Gary Greenfield, Christian Prins, Juan Romero, Giovanni Squillero, Ernesto Tarantino, Andrea G. B. Tettamanzi, Neil Urquhart, and A. Sima Uyar, editors, *Applications of Evolutionary Computing, EvoApplications 2011: EvoCOMNET, EvoFIN, EvoHOT, EvoMUSART, EvoSTIM, EvoTRANSLOG*, volume 6625 of *LNCS*, pages 204–213, Turin, Italy, 27-29 April 2011. Springer Verlag. doi: 10.1007/978-3-642-20520-0_21.

[32] Jonathan Byrne, Erik Hemberg, Michael O'Neill, and Anthony Brabazon. A methodology for user directed search in evolutionary design. *Genetic Programming and Evolvable Machines*, 14(3):287–314, 2013. ISSN 1389-2576. doi: 10.1007/s10710-013-9189-6.

[33] Manuel Cebrian, Manuel Alfonseca, and Alfonso Ortega. Towards the validation of plagiarism detection tools by means of grammar evolution. *IEEE Transactions on Evolutionary Computation*, 13(3):477–485, June 2009. ISSN 1089-778X. doi: 10.1109/TEVC.2008.2008797.

[34] Robert Cleary and Michael O'Neill. An attribute grammar decoder for the 01 multiconstrained knapsack problem. In Günther R. Raidl and Jens Gottlieb, editors, *Evolutionary Computation in Combinatorial Optimization – EvoCOP 2005*, volume 3448 of *LNCS*, pages 34–45, Lausanne, Switzerland, 30 March-1 April 2005. Springer Verlag. doi: 10.1007/978-3-540-31996-2_4.

[35] Wei Cui. *An Empirical Investigation of Price Impact: An Agent-based Modelling Approach.* PhD thesis, University College Dublin, 2012.

[36] Wei Cui, Anthony Brabazon, and Michael O'Neill. Dynamic trade execution: a grammatical evolution approach. *International Journal of Financial Markets and Derivatives*, 2(1):4–31, 2011.

[37] Jamie Cullen. Evolving digital circuits in an industry standard hardware description language. In Xiaodong Li, Michael Kirley, Mengjie Zhang, David G. Green, Victor Ciesielski, Hussein A. Abbass, Zbigniew Michalewicz, Tim Hendtlass, Kalyanmoy Deb, Kay Chen Tan, Jürgen Branke, and Yuhui Shi, editors, *Proceedings of the 7th International Conference on Simulated Evolution And Learning (SEAL '08)*, volume 5361 of *Lecture Notes in Computer Science*, pages 514–523, Melbourne, Australia, December 7-10 2008. Springer. doi: 10.1007/978-3-540-89694-4_52.

[38] Jason M. Daida, Derrick S. Ampy, Michael Ratanasavetavadhana, Hsiaolei Li, and Omar A. Chaudhri. Challenges with verification, repeatability, and meaningful comparison in genetic programming: Gibson's magic. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1851–1858, Orlando, Florida, USA, 1999. Morgan Kaufmann.

[39] Jason M. Daida, Hsiaolei Li, Ricky Tang, and Adam M. Hilss. What makes a problem GP-hard? validating a hypothesis of structural causes. In E. Cantú-Paz, J. A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. Dowsland, N. Jonoska, and J. Miller, editors, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 1665–1677, Chicago, 12-16 July 2003. Springer-Verlag. ISBN 3-540-40603-4. doi: 10.1007/3-540-45110-2_60.

[40] Ian Dempsey, Michael O'Neill, and Anthony Brabazon. Investigations into market index trading models using evolutionary automatic programming. In Michael O'Neill, RichardF.E. Sutcliffe, Conor Ryan, Malachy Eaton, and NiallJ.L. Griffith, editors, *Artificial Intelligence and Cognitive Science*, volume 2464 of *Lecture Notes in Computer Science*, pages 165–170. Springer Berlin Heidelberg, 2002. ISBN 978-3-540-44184-7. doi: 10.1007/3-540-45750-X_21.

[41] Ian Dempsey, Michael O'Neill, and Anthony Brabazon. Adaptive trading with grammatical evolution. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 9137–9142, Vancouver, 6-21 July 2006. IEEE Press. ISBN 0-7803-9487-9. doi: 10.1109/CEC.2006.1688631.

[42] Ian Dempsey, Michael O'Neill, and Anthony Brabazon. *Foundations in Grammatical Evolution for Dynamic Environments*, volume 194 of *Studies in Computational Intelligence*. Springer, April 2009.

[43] Miguel Nuno Fialho dos Santos Nicolau. *Genetic Algorithms using Grammatical Evolution.* PhD thesis, University Of Limerick, 2006.

[44] Jan Drchal and Miroslav Snorek. Grammatical evolution for development of neural networks with real-valued weights using cellular encoding. In *European Simulation and Modelling Conference*, pages 191–195. EUROSIS - ETI, 2008.

[45] Peter Eggenberger. Evolving morphologies of simulated 3d organisms based on differential gene expression. In *Proceedings of the Fourth European Conference on Artificial Life*, pages 205–213. MIT Press, 1997.

[46] Sean Luke *et al.* ECJ evolutionary computation toolkit. Available at http://cs.gmu.edu/∼eclab/projects/ecj/, 1998.

[47] David Fagan, Miguel Nicolau, Michael O'Neill, Edgar Galvan-Lopez, Anthony Brabazon, and Sean McGarraghy. Investigating mapping order in piGE. In *2010 IEEE World Congress on Computational Intelligence*, pages 3058–3064, Barcelona, Spain, 18-23 July 2010. IEEE Computational Intelligence Society, IEEE Press. doi: 10.1109/CEC.2010.5586204.

[48] David Fagan, Michael O'Neill, Edgar Galvan-Lopez, Anthony Brabazon, and Sean McGarraghy. An analysis of genotype-phenotype maps in grammatical evolution. In Anna Isabel Esparcia-Alcazar, Aniko Ekart, Sara Silva, Stephen Dignum, and A. Sima Uyar, editors, *Proceedings of the 13th European Conference on Genetic Programming, EuroGP 2010*, volume 6021 of *LNCS*, pages 62–73, Istanbul, 7-9 April 2010. Springer. doi: 10.1007/978-3-642-12148-7_6.

[49] David Fagan, Miguel Nicolau, Erik Hemberg, Michael O'Neill, Anthony Brabazon, and Sean McGarraghy. Investigation of the performance of different mapping orders for GE on the max problem. In Sara Silva, James A. Foster, Miguel Nicolau, Mario Giacobini, and Penousal Machado, editors, *Proceedings of the 14th European Conference on Genetic Programming, EuroGP 2011*, volume 6621 of *LNCS*, pages 286–297, Turin, Italy, 27-29 April 2011. Springer Verlag. doi: 10.1007/978-3-642-20407-4_25.

[50] David Fagan, Erik Hemberg, Miguel Nicolau, Michael O'Neill, and Sean McGarraghy. Towards adaptive mutation in grammatical evolution. In Terry Soule, Anne Auger, Jason Moore, David Pelta, Christine Solnon, Mike Preuss, Alan Dorin, Yew-Soon Ong, Christian Blum, Dario Landa Silva, Frank Neumann, Tina Yu, Aniko Ekart, Will Browne, Tim Kovacs, Man-Leung Wong, Clara Pizzuti, Jon Rowe, Tobias Friedrich, Giovanni Squillero, Nicolas Bredeche, Stephen Smith, Alison Motsinger-Reif, Jose Lozano, Martin Pelikan, Silja Meyer-Nienberg, Christian Igel, Greg Hornby, Rene Doursat, Steve Gustafson, Gustavo Olague, Shin Yoo, John Clark, Gabriela Ochoa, Gisele Pappa, Fernando Lobo, Daniel Tauritz, Jurgen Branke, and Kalyanmoy Deb, editors, *GECCO Companion '12: Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, pages 1481–1482, Philadelphia, Pennsylvania, USA, 7-11 July 2012. ACM. doi: 10.1145/2330784.2331002.

[51] David Fagan, Erik Hemberg, Michael O'Neill, and Sean McGarraghy. Fitness reactive mutation in grammatical evolution. In Radomil Matousek, editor, *18th International Conference on Soft Computing, MENDEL 2012*, pages 144–149, Brno, Czech Republic, 27-29 June 2012. Brno University of Technology.

[52] David Fagan, Erik Hemberg, Michael O'Neill, and Sean McGarraghy. Understanding expansion order and phenotypic connectivity in piGE. In Krzysztof Krawiec, Alberto Moraglio, Ting Hu, A. Sima Uyar, and Bin Hu, editors, *Proceedings of the 16th European Conference on Genetic Programming, EuroGP 2013*, volume 7831 of *LNCS*, pages 37–48, Vienna, Austria, 3-5 April 2013. Springer Verlag. doi: 10.1007/978-3-642-37207-0_4.

[53] Lawrence J. Fogel. *Intelligence through simulated evolution: forty years of evolutionary programming.* John Wiley & Sons, Inc., New York, NY, USA, 1999. ISBN 0-471-33250-X.

[54] Edgar Galvan-Lopez, David Fagan, Eoin Murphy, John Mark Swafford, Alexandros Agapitos, Michael O'Neill, and Anthony Brabazon. Comparing the performance of the evolvable PiGrammatical evolution genotype-phenotype map to grammatical evolution in the dynamic Ms. Pac-Man environment. In *2010 IEEE World Congress on Computational Intelligence*, pages 1587–1594, Barcelona, Spain, 18-23 July 2010. IEEE Computational Intelligence Society, IEEE Press. doi: 10.1109/CEC.2010.5586508.

[55] Edgar Galvan-Lopez, John Mark Swafford, Michael O'Neill, and Anthony Brabazon. Evolving a Ms. PacMan controller using grammatical evolution. In Cecilia Di Chio, Stefano Cagnoni, Carlos Cotta, Marc Ebner, Aniko Ekart, Anna I. Esparcia-Alcazar, Chi-Keong Goh, Juan J. Merelo, Ferrante Neri, Mike Preuss, Julian Togelius, and Georgios N. Yannakakis, editors, *EvoGAMES*, volume 6024 of *LNCS*, pages 161–170, Istanbul, 7-9 April 2010. Springer. doi: 10.1007/978-3-642-12239-2_17.

[56] George Georgoulas, Dimitris Gavrilis, Ioannis G Tsoulos, Chrysostomos Stylios, Joao Bernardes, and Peter P Groumpos. Novel approach for fetal heart rate classification introducing grammatical evolution. *Biomedical Signal Processing and Control*, 2(2): 69–79, 2007.

[57] Andreas Geyer-Schulz. *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning*, volume 3 of *Studies in Fuzziness.* Physica-Verlag, Heidelberg, 1995. ISBN 3-7908-0830-X.

[58] Scott F. Gilbert. *Developmental Biology.* Sinauer Associates Inc, 8 edition, 2006.

[59] David E. Goldberg. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms.* Kluwer Academic Publishers, Norwell, MA, USA, 2002. ISBN 1402070985.

[60] Jennifer Hallinan and Janet Wiles. Evolving genetic regulatory networks using an artificial genome. In *SECOND ASIA-PACIFIC BIOINFORMATICS CONFERENCE (APBC2004). VOLUME 29 OF CRPIT., DUNEDIN, NEW ZEALAND, ACS (2004) 291*, pages 291–296, 2004.

[61] Hoang Tuan Hao, Nguyen Xuan Hoai, and Robert I McKay. Does it matter where you start? A comparison of two initialisation strategies for grammar guided genetic programming. In R I McKay and Sung-Bae Cho, editors, *Proceedings of The Second Asian-Pacific Workshop on Genetic Programming*, Cairns, Australia, 6-7 December 2004.

[62] Simon Harding, Julian F. Miller, and Wolfgang Banzhaf. SMCGP2: self modifying cartesian genetic programming in two dimensions. In *GECCO '11: Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1491–1498, Dublin, Ireland, 12-16 July 2011. ACM. ISBN 978-1-4503-0557-0. doi: 10.1145/2001576.2001777.

[63] Robin Harper. GE, explosive grammars and the lasting legacy of bad initialisation. In *IEEE Congress on Evolutionary Computation (CEC 2010)*, Barcelona, Spain, 18-23 July 2010. IEEE Press.

[64] Robin Harper. Co-evolving robocode tanks. In Natalio Krasnogor, Pier Luca Lanzi, Andries Engelbrecht, David Pelta, Carlos Gershenson, Giovanni Squillero, Alex Freitas, Marylyn Ritchie, Mike Preuss, Christian Gagne, Yew Soon Ong, Guenther Raidl, Marcus Gallager, Jose Lozano, Carlos Coello-Coello, Dario Landa Silva, Nikolaus Hansen, Silja Meyer-Nieberg, Jim Smith, Gus Eiben, Ester Bernado-Mansilla, Will Browne, Lee Spector, Tina Yu, Jeff Clune, Greg Hornby, Man-Leung Wong, Pierre Collet, Steve Gustafson, Jean-Paul Watson, Moshe Sipper, Simon Poulding, Gabriela Ochoa, Marc Schoenauer, Carsten Witt, and Anne Auger, editors, *GECCO '11: Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1443–1450, Dublin, Ireland, 12-16 July 2011. ACM. ISBN 978-1-4503-0557-0. doi: 10.1145/2001576.2001770.

[65] Robin Harper. Dynamic L-systems in GE. In Natalio Krasnogor, Pier Luca Lanzi, Andries Engelbrecht, David Pelta, Carlos Gershenson, Giovanni Squillero, Alex Freitas, Marylyn Ritchie, Mike Preuss, Christian Gagne, Yew Soon Ong, Guenther Raidl, Marcus Gallager, Jose Lozano, Carlos Coello-Coello, Dario Landa Silva, Nikolaus Hansen, Silja Meyer-Nieberg, Jim Smith, Gus Eiben, Ester Bernado-Mansilla, Will Browne, Lee Spector, Tina Yu, Jeff Clune, Greg Hornby, Man-Leung Wong, Pierre Collet, Steve Gustafson, Jean-Paul Watson, Moshe Sipper, Simon Poulding, Gabriela Ochoa, Marc Schoenauer, Carsten Witt, and Anne Auger, editors, *GECCO '11: Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, pages 209–210, Dublin, Ireland, 12-16 July 2011. ACM. doi: 10.1145/2001858.2001975.

[66] Robin Harper and Alan Blair. A structure preserving crossover in grammatical evolution. In David Corne, Zbigniew Michalewicz, Marco Dorigo, Gusz Eiben, David Fogel, Carlos Fonseca, Garrison Greenwood, Tan Kay Chen, Guenther Raidl, Ali Zalzala, Simon Lucas, Ben Paechter, Jennifier Willies, Juan J. Merelo Guervos, Eugene

Eberbach, Bob McKay, Alastair Channon, Ashutosh Tiwari, L. Gwenn Volkert, Dan Ashlock, and Marc Schoenauer, editors, *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, volume 3, pages 2537–2544, Edinburgh, UK, 2-5 September 2005. IEEE Press. ISBN 0-7803-9363-5. doi: 10.1109/CEC.2005.1555012.

[67] Robin Harper and Alan Blair. Dynamically defined functions in grammatical evolution. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 9188–9195, Vancouver, 6-21 July 2006. IEEE Press. ISBN 0-7803-9487-9. doi: 10.1109/CEC.2006.1688638.

[68] Robin Thomas Ross Harper. *Enhancing Grammatical Evolution*. PhD thesis, School of Computer Science and Engineering, The University of New South Wales, Sydney 2052, Australia, 2009.

[69] M. A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1978. ISBN 0201029553.

[70] Erik Hemberg. *An Exploration of Grammars in Grammatical Evolution*. PhD thesis, University College Dublin, 2010.

[71] Erik Hemberg and James McDermott *et al.* PonyGE: a pony-sized implementation of grammatical evolution in python. Available at https://code.google.com/p/ponyge, 2010.

[72] Erik Hemberg, Michael O'Neill, and Anthony Brabazon. An investigation into automatically defined function representations in grammatical evolution. In R. Matousek and L. Nolle, editors, *15th International Conference on Soft Computing, Mendel'09*, Brno, Czech Republic, 24-26 June 2009.

[73] Erik Hemberg, Lester Ho, Michael O'Neill, and Holger Claussen. A comparison of grammatical genetic programming grammars for controlling femtocell network coverage. *Genetic Programming and Evolvable Machines*, 14(1):65–93, March 2013. ISSN 1389-2576. doi: 10.1007/s10710-012-9171-8.

[74] Martin Hemberg and Una-May O'Reilly. Extending grammatical evolution to evolve digital surfaces with genr8. In Maarten Keijzer, Una-May O'Reilly, Simon M. Lucas, Ernesto Costa, and Terence Soule, editors, *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, volume 3003 of *LNCS*, pages 299–308, Coimbra, Portugal, 5-7 April 2004. Springer-Verlag. ISBN 3-540-21346-5.

[75] N. X. Hoai. Solving the symbolic regression with tree-adjunct grammar guided genetic programming: The preliminary results. In Nikola Kasabov and Peter Whigham, editors, *Australasia-Japan Workshop on Intelligent and Evolutionary Systems*, University of Otago, Dunedin, New Zealand, 19-21st November 2001.

[76] N. X. Hoai. Solving trignometric identities with tree adjunct grammar guided genetic programming. In Ajith Abraham and Mario Koppen, editors, *2001 International Workshop on Hybrid Intelligent Systems*, LNCS, pages 339–352, Adelaide, Australia, 11-12 December 2001. Springer-Verlag. ISBN 3-7908-1480-6.

[77] N. X. Hoai. Solving the symbolic regression problem with tree-adjunct grammar guided genetic programming: The preliminary results. In *In Proceedings of Congress on Evolutionary Computation (CEC'2002), Hawai*, pages 1326–1331. IEEE Press, 2002.

[78] Nguyen Xuan Hoai. *A Flexible Representation for Genetic Programming from Natural Language Processing*. PhD thesis, Australian Defence force Academy, University of New South Wales, Australia, December 2004.

[79] Nguyen Xuan Hoai and R. I. McKay. Softening the structural difficulty in genetic programming with TAG-based representation and insertion/deletion operators. In Kalyanmoy Deb, Riccardo Poli, Wolfgang Banzhaf, Hans-Georg Beyer, Edmund Burke, Paul Darwen, Dipankar Dasgupta, Dario Floreano, James Foster, Mark Harman, Owen Holland, Pier Luca Lanzi, Lee Spector, Andrea Tettamanzi, Dirk Thierens, and Andy Tyrrell, editors, *Genetic and Evolutionary Computation – GECCO-2004, Part II*, volume 3103 of *Lecture Notes in Computer Science*, pages 605–616, Seattle, WA, USA, 26-30 June 2004. Springer-Verlag. ISBN 3-540-22343-6.

[80] Nguyen Xuan Hoai and Robert Ian McKay. An investigation on the roles of insertion and deletion operators in tree adjoining grammar guided genetic programming. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 472–477, Portland, Oregon, 20-23 June 2004. IEEE Press. ISBN 0-7803-8515-2. doi: 10.1109/CEC.2004.1330894.

[81] Nguyen Xuan Hoai, R. I. McKay, and D. Essam. Some experimental results with tree adjunct grammar guided genetic programming. In James A. Foster, Evelyne Lutton, Julian Miller, Conor Ryan, and Andrea G. B. Tettamanzi, editors, *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002*, volume 2278 of *LNCS*, pages 228–237, Kinsale, Ireland, 3-5 April 2002. Springer-Verlag. ISBN 3-540-43378-3. doi: 10.1007/3-540-45984-7_22.

[82] Nguyen Xuan Hoai, R. I. McKay, and H. A. Abbass. Tree adjoining grammars, language bias, and genetic programming. In Conor Ryan, Terence Soule, Maarten Keijzer, Edward Tsang, Riccardo Poli, and Ernesto Costa, editors, *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 335–344, Essex, 14-16 April 2003. Springer-Verlag. ISBN 3-540-00971-X.

[83] Nguyen Xuan Hoai, R. I. (Bob) McKay, Daryl Essam, and Hussein Abbass. Toward an alternative comparison between different genetic programming systems. In Maarten Keijzer, Una-May O'Reilly, Simon M. Lucas, Ernesto Costa, and Terence

Soule, editors, *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, volume 3003 of *LNCS*, pages 67–77, Coimbra, Portugal, 5-7 April 2004. Springer-Verlag. ISBN 3-540-21346-5.

[84] Nguyen Xuan Hoai, Robert I. McKay, Daryl Essam, and Hoang Tuan Hao. Genetic transposition in tree-adjoining grammar guided genetic programming: The duplication operator. In Maarten Keijzer, Andrea Tettamanzi, Pierre Collet, Jano I. van Hemert, and Marco Tomassini, editors, *Proceedings of the 8th European Conference on Genetic Programming*, volume 3447 of *Lecture Notes in Computer Science*, pages 108–119, Lausanne, Switzerland, 30 March - 1 April 2005. Springer. ISBN 3-540-25436-6. doi: 10.1007/b107383.

[85] Nguyen Xuan Hoai, R. I. (Bob) McKay, and Daryl Essam. Representation and structural difficulty in genetic programming. *IEEE Transactions on Evolutionary Computation*, 10(2):157–166, April 2006.

[86] N.X. Hoai and R.I. McKay. A framework for tree adjunct grammar guided genetic programming. In *Proceedings of the Post-graduate ADFA Conference on Computer Science*, pages 93–99. ADFA, 2001. doi: 10.1.1.79.4037.

[87] Tuan-Hao Hoang. *Evolutionary Developmental Evaluation : the Interplay between Evolution and Development.* PhD thesis, Information Technology & Electrical Engineering, Australian Defence Force Academy, University of New South Wales, Australia, December 2008.

[88] Tuan-Hao Hoang, Daryl Essam, R. I. McKay, and Xuan Hoai Nguyen. Developmental evaluation in genetic programming: A TAG-based framework. In The Long Pham, Hai Khoi Le, and Xuan Hoai Nguyen, editors, *Proceedings of the Third Asian-Pacific workshop on Genetic Programming*, pages 86–97, Military Technical Academy, Hanoi, VietNam, 2006.

[89] Tuan-Hao Hoang, Daryl Essam, R. I. (Bob) McKay, and Xuan Hoai Nguyen. Solving symbolic regression problems using incremental evaluation in genetic programming. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 7487–7494, Vancouver, 6-21 July 2006. IEEE Press. ISBN 0-7803-9487-9. doi: 10.1109/CEC.2006.1688570.

[90] Tuan-Hao Hoang, R. McKay, D. Essam, and Xuan Hoai Nguyen. Developmental evaluation in genetic programming: A position paper. In *Frontiers in the Convergence of Bioscience and Information Technologies, FBIT 2007*, pages 773–778, Jeju City, Korea, 11-13 October 2007. IEEE Press. doi: 10.1109/FBIT.2007.104.

[91] Tuan-Hao Hoang, Daryl Essam, R. I. (Bob) McKay, and Nguyen Xuan Hoai. Developmental evaluation in genetic programming: The TAG-based frame work. *International Journal of Knowledge-Based and Intelligent Engineering Systems*, 12(1): 69–82, 2008. ISSN 1327-2314.

[92] Tuan Hao Hoang, R. I. (Bob) McKay, Daryl Essam, and Nguyen Xuan Hoai. Learning general solutions through multiple evaluations during development. In Gregory Hornby, Lukás Sekanina, and Pauline C. Haddow, editors, *Proceedings of the 8th International Conference Evolvable Systems: From Biology to Hardware, ICES 2008*, volume 5216 of *Lecture Notes in Computer Science*, pages 201–212, Prague, Czech Republic, September 21-24 2008. Springer. doi: 10.1007/978-3-540-85857-7_18.

[93] Tuan-Hao Hoang, R. I. McKay, Daryl Essam, and Nguyen Xuan Hoai. On synergistic interactions between evolution, development and layered learning. *IEEE Transactions on Evolutionary Computation*, 15(3):287–312, June 2011. ISSN 1089-778X. doi: 10.1109/TEVC.2011.2150752.

[94] Jonatan Hugosson, Erik Hemberg, Anthony Brabazon, and Michael O'Neill. Genotype representations in grammatical evolution. *Applied Soft Computing*, 10(1):36–43, January 2010. doi: 10.1016/j.asoc.2009.05.003.

[95] Terry Jones. *Evolutionary Algorithms, Fitness Landscapes, and Search.* PhD thesis, University of New Mexico, 1995.

[96] A.K. Joshi. *Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions*, chapter 6, pages 205–250. Cambridge University Press, New York, 1985.

[97] A.K. Joshi, L.S. Levy, and M. Takahashi. Tree Adjunct Grammars. *Journal of Computer and System Sciences*, 10(1):136–163, 1975. ISSN 00220000. doi: 10.1016/S0022-0000(75)80019-5.

[98] Aravind K Joshi and Yves Schabes. Tree-adjoining grammars and lexicalized grammars, 1991.

[99] Aravind K. Joshi and Yves Schabes. *Tree-Adjoining Grammars*, pages 69–123. Springer Berlin Heidelberg, 1997. ISBN 978-3-642-63859-6. doi: 10.1007/978-3-642-59126-6_2.

[100] Moonyoung Kang, Jungseok Shin, Tuan Hao Hoang, R. I. (Bob) McKay, Daryl Essam, Naoki Mori, and Xuan Hoai Nguyen. Code duplication and developmental evaluation in genetic programming. In *Proceedings of the 2006 Asia-Pacific Workshop on Intelligent and Evolutionary Systems*, pages 181–191, Seoul, Korea, November 2006.

[101] Stuart A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution.* Oxford University Press, USA, 1 edition, June 1993. ISBN 0195079515.

[102] M. Keijzer, V. Babovic, C. Ryan, M. O'Neill, and M. Cattolico. Adaptive logic programming. In Lee Spector, Erik D. Goodman, Annie Wu, W. B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary*

Computation Conference (GECCO-2001), pages 42–49, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann. ISBN 1-55860-774-9.

[103] Maarten Keijzer, Conor Ryan, Michael O'Neill, Mike Cattolico, and Vladan Babovic. Ripple crossover in genetic programming. In Julian F. Miller, Marco Tomassini, Pier Luca Lanzi, Conor Ryan, Andrea G. B. Tettamanzi, and William B. Langdon, editors, *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038 of *LNCS*, pages 74–86, Lake Como, Italy, 18-20 April 2001. Springer-Verlag. ISBN 3-540-41899-7.

[104] Maarten Keijzer, Michael O'Neill, Conor Ryan, and Mike Cattolico. Grammatical evolution rules: The mod and the bucket rule. In James A. Foster, Evelyne Lutton, Julian Miller, Conor Ryan, and Andrea G. B. Tettamanzi, editors, *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002*, volume 2278 of *LNCS*, pages 123–130, Kinsale, Ireland, 3-5 April 2002. Springer-Verlag. ISBN 3-540-43378-3. doi: 10.1007/3-540-45984-7_12.

[105] Robert E. Keller and Wolfgang Banzhaf. Genetic programming using genotype-phenotype mapping from linear genomes into linear phenotypes. In *Proceedings of the First Annual Conference on Genetic Programming*, GECCO '96, pages 116–122, Cambridge, MA, USA, 1996. MIT Press. ISBN 0-262-61127-9.

[106] Maryam Mahsal Khan, Gul Muhammad Khan, and Julian F. Miller. Evolution of neural networks using cartesian genetic programming. In *IEEE Congress on Evolutionary Computation (CEC 2010)*, Barcelona, Spain, 18-23 July 2010. IEEE Press. doi: 10.1109/CEC.2010.5586547.

[107] MinHyeok Kim, Robert Ian (Bob) McKay, Nguyen Xuan Hoai, and Kangil Kim. Operator self-adaptation in genetic programming. In Sara Silva, James A. Foster, Miguel Nicolau, Mario Giacobini, and Penousal Machado, editors, *Proceedings of the 14th European Conference on Genetic Programming, EuroGP 2011*, volume 6621 of *LNCS*, pages 215–226, Turin, Italy, 27-29 April 2011. Springer Verlag. doi: 10.1007/978-3-642-20407-4_19.

[108] MinHyeok Kim, Bob McKay, Kangil Kim, and Xuan Hoai Nguyen. Analysing the effects of diverse operators in a genetic programming system. In Carlos A. Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone, editors, *Parallel Problem Solving from Nature, PPSN XII (part 1)*, volume 7491 of *Lecture Notes in Computer Science*, pages 387–396, Taormina, Italy, September 1-5 2012. Springer. doi: 10.1007/978-3-642-32937-1_39.

[109] MinHyeok Kim, Robert Ian (Bob) McKay, Dong-Kyun Kim, and Xuan Hoai Nguyen. Evolutionary operator self-adaptation with diverse operators. In Alberto Moraglio, Sara Silva, Krzysztof Krawiec, Penousal Machado, and Carlos Cotta, editors, *Proceedings of the 15th European Conference on Genetic Programming, EuroGP 2012*,

volume 7244 of *LNCS*, pages 230–241, Malaga, Spain, 11-13 April 2012. Springer Verlag. doi: 10.1007/978-3-642-29139-5_20.

[110] John R. Koza and Martin A. Keane. Cart centering and broom balancing by genetically breeding populations of control strategy programs. In *Proceedings of International Joint Conference on Neural Networks*, volume I, pages 198–201, Washington, 15-19 January 1990. Lawrence Erlbaum.

[111] John R. Koza and Riccardo Poli. Genetic programming. In *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, chapter 5, pages 127–164. Springer, 2005.

[112] JohnR. Koza. Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines*, 11(3-4):251–284, 2010. ISSN 1389-2576. doi: 10.1007/s10710-010-9112-3.

[113] J.R. Koza. *Genetic Programming.* MIT Press, 1992. ISBN 0-262-11170-5.

[114] A. Kroch and A.K. Joshi. The Linguistic Relevance of Tree Adjoining Grammar, 1985.

[115] Jean Krohn and Denise Gorse. Fractal gene regulatory networks for control of nonlinear systems. In Robert Schaefer, Carlos Cotta, Joanna Koodziej, and Gnter Rudolph, editors, *Parallel Problem Solving from Nature, PPSN XI*, volume 6239 of *Lecture Notes in Computer Science*, pages 209–218. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-15870-4. doi: 10.1007/978-3-642-15871-1_22.

[116] Jean Krohn and Denise Gorse. Extracting key gene regulatory dynamics for the direct control of mechanical systems. In CarlosA.Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone, editors, *Parallel Problem Solving from Nature - PPSN XII*, volume 7491 of *Lecture Notes in Computer Science*, pages 468–477. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-32936-4. doi: 10.1007/978-3-642-32937-1_47.

[117] Jean Krohn, Peter J. Bentley, and Hooman Shayani. The challenge of irrationality: fractal protein recipes for pi. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 715–722, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-325-9. doi: 10.1145/1569901.1570000.

[118] P. Dwight Kuo, Andre Leier, and Wolfgang Banzhaf. Evolving dynamics in an artificial regulatory network model. In *Proc. of the Paralell Problem Solving from Nature Conference, volume LNCS 3242*, pages 571–580. Springer, 2004.

[119] P. Dwight Kuo, Wolfgang Banzhaf, and Andre Leier. 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, 2006. ISSN 0303-2647. doi: DOI:10.1016/j.biosystems.2006.01.004.

[120] W. B. Langdon and R. Poli. Why ants are hard. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 193–201, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann. ISBN 1-55860-548-7.

[121] Rui Lopes and Ernesto Costa. ReNCode: A regulatory network computational device. In Sara Silva, James A. Foster, Miguel Nicolau, Mario Giacobini, and Penousal Machado, editors, *Proceedings of the 14th European Conference on Genetic Programming, EuroGP 2011*, volume 6621 of *LNCS*, pages 142–153, Turin, Italy, 27-29 April 2011. Springer Verlag. doi: doi:10.1007/978-3-642-20407-4_13.

[122] Rui L. Lopes and Ernesto Costa. Using feedback in a regulatory network computational device. In Natalio Krasnogor, Pier Luca Lanzi, Andries Engelbrecht, David Pelta, Carlos Gershenson, Giovanni Squillero, Alex Freitas, Marylyn Ritchie, Mike Preuss, Christian Gagne, Yew Soon Ong, Guenther Raidl, Marcus Gallager, Jose Lozano, Carlos Coello-Coello, Dario Landa Silva, Nikolaus Hansen, Silja Meyer-Nieberg, Jim Smith, Gus Eiben, Ester Bernado-Mansilla, Will Browne, Lee Spector, Tina Yu, Jeff Clune, Greg Hornby, Man-Leung Wong, Pierre Collet, Steve Gustafson, Jean-Paul Watson, Moshe Sipper, Simon Poulding, Gabriela Ochoa, Marc Schoenauer, Carsten Witt, and Anne Auger, editors, *GECCO '11: Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1499–1506, Dublin, Ireland, 12-16 July 2011. ACM. doi: doi:10.1145/2001576.2001778.

[123] Rui L. Lopes and Ernesto Costa. The regulatory network computational device. *Genetic Programming and Evolvable Machines*, 13(3):339–375, September 2012. ISSN 1389-2576. doi: doi:10.1007/s10710-012-9160-y. Special issue on selected papers from the 2011 European conference on genetic programming.

[124] Rui L. Lopes and Ernesto Costa. Genetic programming with genetic regulatory networks. In Christian Blum, Enrique Alba, Anne Auger, Jaume Bacardit, Josh Bongard, Juergen Branke, Nicolas Bredeche, Dimo Brockhoff, Francisco Chicano, Alan Dorin, Rene Doursat, Aniko Ekart, Tobias Friedrich, Mario Giacobini, Mark Harman, Hitoshi Iba, Christian Igel, Thomas Jansen, Tim Kovacs, Taras Kowaliw, Manuel Lopez-Ibanez, Jose A. Lozano, Gabriel Luque, John McCall, Alberto Moraglio, Alison Motsinger-Reif, Frank Neumann, Gabriela Ochoa, Gustavo Olague, Yew-Soon Ong, Michael E. Palmer, Gisele Lobo Pappa, Konstantinos E. Parsopoulos, Thomas Schmickl, Stephen L. Smith, Christine Solnon, Thomas Stuetzle, El-Ghazali Talbi, Daniel Tauritz, and Leonardo Vanneschi, editors, *GECCO '13: Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference*, pages 965–972, Amsterdam, The Netherlands, 6-10 July 2013. ACM. doi: doi:10.1145/2463372.2463488.

[125] Rui L. Lopes and Ernesto Costa. GEARNet: grammatical evolution with artificial regulatory networks. In Christian Blum, Enrique Alba, Anne Auger, Jaume

Bacardit, Josh Bongard, Juergen Branke, Nicolas Bredeche, Dimo Brockhoff, Francisco Chicano, Alan Dorin, Rene Doursat, Aniko Ekart, Tobias Friedrich, Mario Giacobini, Mark Harman, Hitoshi Iba, Christian Igel, Thomas Jansen, Tim Kovacs, Taras Kowaliw, Manuel Lopez-Ibanez, Jose A. Lozano, Gabriel Luque, John McCall, Alberto Moraglio, Alison Motsinger-Reif, Frank Neumann, Gabriela Ochoa, Gustavo Olague, Yew-Soon Ong, Michael E. Palmer, Gisele Lobo Pappa, Konstantinos E. Parsopoulos, Thomas Schmickl, Stephen L. Smith, Christine Solnon, Thomas Stuetzle, El-Ghazali Talbi, Daniel Tauritz, and Leonardo Vanneschi, editors, *GECCO '13: Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference*, pages 973–980, Amsterdam, The Netherlands, 6-10 July 2013. ACM. doi: doi:10.1145/2463372.2463490.

[126] Sean Luke. Two fast tree-creation algorithms for genetic programming. *IEEE Transactions on Evolutionary Computation*, 4(3):274–283, 2000.

[127] Sean Luke. *Essentials of Metaheuristics*. lulu.com, first edition, 2009.

[128] James McDermott, Michael O'Neill, and Niall J. L. Griffith. Interactive ec control of synthesized timbre. *Evol. Comput.*, 18(2):277–303, June 2010. ISSN 1063-6560. doi: 10.1162/evco.2010.18.2.18205.

[129] James McDermott, John Mark Swafford, Martin Hemberg, Jonathan Byrne, Erik Hemberg, Michael Fenton, Ciaran McNally, Elizabeth Shotton, and Michael O'Neill. An assessment of string-rewriting grammars for evolutionary architectural design. *Environment and Planning B*, 39(4):713–731, 2012. doi: 10.1068/b38037.

[130] Seán McGarraghy and Michael Phelan. Generating supply chain ordering policies using quantum inspired genetic algorithms and grammatical evolution. In Georgios Dounias Ioannis Minis, Vasileios Zeimpekis and Nicholas Ampazis, editors, *Supply Chain Optimization, Design, and Management: Advances and Intelligent Methods*, pages 125–154. IGI Global, 2011. doi: 10.4018/978-1-61520-633-9.ch006.

[131] Richard McGee, Michael O'Neill, and Anthony Brabazon. The syntax of stock selection: Grammatical evolution of a stock picking model. In *2010 IEEE World Congress on Computational Intelligence*, pages 4347–4354, Barcelona, Spain, 18-23 July 2010. IEEE Computational Intelligence Society, IEEE Press. doi: 10.1109/CEC.2010.5586001.

[132] Robert I. McKay, Xuan Hoai Nguyen, James R. Cheney, MinHyeok Kim, Naoki Mori, and Tuan Hao Hoang. Estimating the distribution and propagation of genetic programming building blocks through tree compression. In Guenther Raidl, Franz Rothlauf, Giovanni Squillero, Rolf Drechsler, Thomas Stuetzle, Mauro Birattari, Clare Bates Congdon, Martin Middendorf, Christian Blum, Carlos Cotta, Peter Bosman, Joern Grahl, Joshua Knowles, David Corne, Hans-Georg Beyer, Ken Stanley, Julian F. Miller, Jano van Hemert, Tom Lenaerts, Marc Ebner, Jaume Bacardit,

Michael O'Neill, Massimiliano Di Penta, Benjamin Doerr, Thomas Jansen, Riccardo Poli, and Enrique Alba, editors, *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1011–1018, Montreal, 8-12 July 2009. ACM. doi: 10.1145/1569901.1570038.

[133] Robert I. McKay, Nguyen Xuan Hoai, Peter Alexander Whigham, Yin Shan, and Michael O'Neill. Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines*, 11(3/4):365–396, September 2010. ISSN 1389-2576. doi: 10.1007/s10710-010-9109-y. Tenth Anniversary Issue: Progress in Genetic Programming and Evolvable Machines.

[134] Robert I. McKay, Nguyen Xuan Hoai, Peter Alexander Whigham, Yin Shan, and Michael O'Neill. Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines*, 11(3-4):365–396, September 2010. ISSN 1389-2576. doi: 10.1007/s10710-010-9109-y.

[135] Robert Ian McKay, Tuan Hao Hoang, Daryl Leslie Essam, and Xuan Hoai Nguyen. Developmental evaluation in genetic programming: the preliminary results. In Pierre Collet, Marco Tomassini, Marc Ebner, Steven Gustafson, and Anikó Ekárt, editors, *Proceedings of the 9th European Conference on Genetic Programming*, volume 3905 of *Lecture Notes in Computer Science*, pages 280–289, Budapest, Hungary, 10 - 12 April 2006. Springer. ISBN 3-540-33143-3.

[136] Robert Ian (Bob) McKay, Jungseok Shin, Tuan Hao Hoang, Xuan Hoai Nguyen, and Naoki Mori. Using compression to understand the distribution of building blocks in genetic programming populations. In Dipti Srinivasan and Lipo Wang, editors, *2007 IEEE Congress on Evolutionary Computation*, pages 2501–2508, Singapore, 25-28 September 2007. IEEE Computational Intelligence Society, IEEE Press. ISBN 1-4244-1340-0. doi: 10.1109/CEC.2007.4424785.

[137] Brad L. Miller, Brad L. Miller, David E. Goldberg, and David E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9:193–212, 1995.

[138] Alison A Motsinger, David M Reif, Scott M Dudek, and Marylyn D Ritchie. Understanding the evolutionary process of grammatical evolution neural networks for feature selection in genetic epidemiology. In *Computational Intelligence and Bioinformatics and Computational Biology, 2006. CIBCB'06. 2006 IEEE Symposium on*, pages 1–8. IEEE, 2006.

[139] Eoin Murphy. Examining grammars and grammatical evolution in dynamic environments. In Miguel Nicolau, editor, *GECCO 2011 Graduate students workshop*, pages 779–782, Dublin, Ireland, 12-16 July 2011. ACM. doi: 10.1145/2001858.2002090.

[140] Eoin Murphy, Michael O'Neill, Edgar Galvan-Lopez, and Anthony Brabazon. Tree-adjunct grammatical evolution. In *2010 IEEE World Congress on Computational*

Intelligence, pages 4449–4456, Barcelona, Spain, 18-23 July 2010. IEEE Computational Intelligence Society, IEEE Press. doi: 10.1109/CEC.2010.5586497.

[141] Eoin Murphy, Michael O'Neill, and Anthony Brabazon. A comparison of GE and TAGE in dynamic environments. In Natalio Krasnogor, Pier Luca Lanzi, Andries Engelbrecht, David Pelta, Carlos Gershenson, Giovanni Squillero, Alex Freitas, Marylyn Ritchie, Mike Preuss, Christian Gagne, Yew Soon Ong, Guenther Raidl, Marcus Gallager, Jose Lozano, Carlos Coello-Coello, Dario Landa Silva, Nikolaus Hansen, Silja Meyer-Nieberg, Jim Smith, Gus Eiben, Ester Bernado-Mansilla, Will Browne, Lee Spector, Tina Yu, Jeff Clune, Greg Hornby, Man-Leung Wong, Pierre Collet, Steve Gustafson, Jean-Paul Watson, Moshe Sipper, Simon Poulding, Gabriela Ochoa, Marc Schoenauer, Carsten Witt, and Anne Auger, editors, *GECCO '11: Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1387–1394, Dublin, Ireland, 12-16 July 2011. ACM. doi: 10.1145/2001576.2001763.

[142] Eoin Murphy, Michael O'Neill, and Anthony Brabazon. Examining mutation landscapes in grammar based genetic programming. In Sara Silva, James A. Foster, Miguel Nicolau, Mario Giacobini, and Penousal Machado, editors, *Proceedings of the 14th European Conference on Genetic Programming, EuroGP 2011*, volume 6621 of *LNCS*, pages 130–141, Turin, Italy, 27-29 April 2011. Springer Verlag. doi: 10.1007/978-3-642-20407-4_12. Best paper.

[143] Eoin Murphy, Erik Hemberg, Miguel Nicolau, Michael O'Neill, and Anthony Brabazon. Grammar bias and initialisation in grammar based genetic programming. In Alberto Moraglio, Sara Silva, Krzysztof Krawiec, Penousal Machado, and Carlos Cotta, editors, *Proceedings of the 15th European Conference on Genetic Programming, EuroGP 2012*, volume 7244 of *LNCS*, pages 85–96, Malaga, Spain, 11-13 April 2012. Springer Verlag. doi: 10.1007/978-3-642-29139-5_8.

[144] Eoin Murphy, Miguel Nicolau, Erik Hemberg, Michael O'Neill, and Anthony Brabazon. Differential gene expression with tree-adjunct grammars. In Carlos A. Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone, editors, *Parallel Problem Solving from Nature, PPSN XII (part 1)*, volume 7491 of *Lecture Notes in Computer Science*, pages 377–386, Taormina, Italy, September 1-5 2012. Springer. doi: 10.1007/978-3-642-32937-1_38.

[145] James E. Murphy. *Applications of Evolutionary Computation to Quadrupedal Animal Animation*. PhD thesis, School of Computer Science and Informatics, University College Dublin, Ireland, March 2011.

[146] Quang Uy Nguyen, Eoin Murphy, Michael O'Neill, and Xuan Hoai Nguyen. Semantic-based subtree crossover applied to dynamic problems. In Tu Bao Ho, R. I. McKay, Xuan Hoai Nguyen, and The Duy Bui, editors, *The Third International Conference on Knowledge and Systems Engineering, KSE'2011*, pages 78–84, Hanoi University, 14–16 October 2011. IEEE. doi: 10.1109/KSE.2011.20.

[147] X. H. Nguyen, R. I. (Bob) McKay, and D. L. Essam. Solving the symbolic regression problem with tree-adjunct grammar guided genetic programming: The comparative results. *The Australian Journal of Intelligent Information Processing Systems*, 7 (3/4):114–121, 2001.

[148] X. H. Nguyen, R. I. (Bob) McKay, and D. L. Essam. Can tree adjunct grammar guided genetic programming be good at finding a needle in a haystack? A case study. In *IEEE International Conference on Communications, Circuits and Systems*, volume 2, pages 1113–1117, Chengdu, China, July 2002. IEEE Press.

[149] X. H. Nguyen, R. I. (Bob) McKay, and D. L. Essam. Finding trigonometric identities with tree adjunct grammar guided genetic programming. In A. Abraham, L. Jain, and B. J. van der Zwaag, editors, *Innovations in Intelligent Systems and Applications*, volume 140 of *Springer Studies in Fuzziness and Soft Computing*, pages 221–236. Springer-Verlag, Berlin, Germany, January 2004. ISBN 3-540-20265-X.

[150] Xuan Hoai Nguyen, R. I. (Bob) McKay, D. L. Essam, and H. A. Abbass. Genetic transposition in tree-adjoining grammar guided genetic programming: the relocation operator. In *2004 Asia-Pacific Conference on Simulated Evolution and Learning*, Busan, Korea, October 2004.

[151] Miguel Nicolau and Dan Costelloe. Using grammatical evolution to parameterise interactive 3D image generation. In Cecilia Di Chio, Anthony Brabazon, Gianni Di Caro, Rolf Drechsler, Marc Ebner, Muddassar Farooq, Joern Grahl, Gary Greenfield, Christian Prins, Juan Romero, Giovanni Squillero, Ernesto Tarantino, Andrea G. B. Tettamanzi, Neil Urquhart, and A. Sima Uyar, editors, *Applications of Evolutionary Computing, EvoApplications 2011: EvoCOMNET, EvoFIN, EvoHOT, EvoMUSART, EvoSTIM, EvoTRANSLOG*, volume 6625 of *LNCS*, pages 374–383, Turin, Italy, 27-29 April 2011. Springer Verlag. doi: 10.1007/978-3-642-20520-0_38.

[152] Miguel Nicolau and Ian Dempsey. Introducing grammar based extensions for grammatical evolution. In Gary G. Yen, Lipo Wang, Piero Bonissone, and Simon M. Lucas, editors, *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 2663–2670, Vancouver, 6-21 July 2006. IEEE Press. ISBN 0-7803-9487-9. doi: 10.1109/CEC.2006.1688372.

[153] Miguel Nicolau and Conor Ryan. Solving sudoku with the GAuGE system. In Pierre Collet, Marco Tomassini, Marc Ebner, Steven Gustafson, and Anikó Ekárt, editors, *Proceedings of the 9th European Conference on Genetic Programming*, volume 3905 of *Lecture Notes in Computer Science*, pages 213–224, Budapest, Hungary, 10 - 12 April 2006. Springer. ISBN 3-540-33143-3.

[154] Miguel Nicolau and Marc Schoenauer. On the evolution of scale-free topologies with a gene regulatory network model. *Biosystems*, 98(3):137–148, 2009. doi: 10.1016/j.biosystems.2009.06.006.

[155] Miguel Nicolau and Darwin Slattery. *libGE*, 2006. http://bds.ul.ie/libGE/libGE/.

[156] Miguel Nicolau, Marc Schoenauer, and Wolfgang Banzhaf. Evolving genes to balance a pole. In *Proceedings of the 13th European Conference on Genetic Programming, EuroGP 2010*, volume 6021 of *LNCS*, pages 196–207, Istanbul, 7-9 April 2010. Springer.

[157] Miguel Nicolau, Michael O'Neill, and Anthony Brabazon. Termination in grammatical evolution: Grammar design, wrapping, and tails. In Xiaodong Li, editor, *Proceedings of the 2012 IEEE Congress on Evolutionary Computation*, pages 2381–2388, Brisbane, Australia, 10-15 June 2012. ISBN 0-7803-8515-2. doi: 10.1109/CEC.2012.6256563.

[158] Miguel Nicolau, Michael O'Neill, and Anthony Brabazon. Applying genetic regulatory networks to index trading. In Carlos A. Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone, editors, *Parallel Problem Solving from Nature, PPSN XII (part 2)*, volume 7492 of *Lecture Notes in Computer Science*, pages 428–437, Taormina, Italy, September 1-5 2012. Springer. doi: 10.1007/978-3-642-32964-7_43.

[159] Miguel Nicolau, Matthew Saunders, Michael O'Neill, Bruce Osborne, and Anthony Brabazon. Evolving interpolating models of net ecosystem CO2 exchange using grammatical evolution. In Alberto Moraglio, Sara Silva, Krzysztof Krawiec, Penousal Machado, and Carlos Cotta, editors, *Proceedings of the 15th European Conference on Genetic Programming, EuroGP 2012*, volume 7244 of *LNCS*, pages 134–145, Malaga, Spain, 11-13 April 2012. Springer Verlag. doi: 10.1007/978-3-642-29139-5_12.

[160] Nils J. Nilsson. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill Pub. Co., 1971.

[161] Cristian Nuez and Derek Watt. An investigation into the effect of heuristic mappers on the max problem in grammatical evolution. Master's thesis, University College Dublin, 2012.

[162] M. O'Neill, A. Brabazon, M. Nicolau, S.M. Garraghy, and P. Keenan. πGrammatical Evolution. In *Genetic and Evolutionary Computation GECCO 2004*, pages 617–629. Springer Berlin / Heidelberg, 2004.

[163] Michael O'Neill. *Automatic Programming in an Arbitrary Language: Evolving Programs with Grammatical Evolution*. PhD thesis, University of Limerick, 2001.

[164] Michael O'Neill and Anthony Brabazon. Grammatical swarm. In *Genetic and Evolutionary Computation–GECCO 2004*, pages 163–174. Springer, 2004.

[165] Michael O'Neill and Anthony Brabazon. mGGA: The meta-grammar genetic algorithm. In Maarten Keijzer, Andrea Tettamanzi, Pierre Collet, Jano I. van Hemert,

and Marco Tomassini, editors, *Proceedings of the 8th European Conference on Genetic Programming*, volume 3447 of *Lecture Notes in Computer Science*, pages 311–320, Lausanne, Switzerland, 30 March - 1 April 2005. Springer. ISBN 3-540-25436-6. doi: doi:10.1007/b107383.

[166] Michael O'Neill and Anthony Brabazon. Grammatical differential evolution. In Hamid R. Arabnia, editor, *Proceedings of the 2006 International Conference on Artificial Intelligence, ICAI 2006*, volume 1, pages 231–236, Las Vegas, Nevada, USA, June 26-29 2006. CSREA Press. ISBN 1-932415-96-3.

[167] Michael O'Neill and Anthony Brabazon. Grammatical swarm: The generation of programs by social programming. *Natural Computing*, 5(4):443–462, 2006.

[168] Michael O'Neill and Anthony Brabazon. Evolving a logo design using lindenmayer systems, postscript and grammatical evolution. In Jun Wang, editor, *2008 IEEE World Congress on Computational Intelligence*, pages 3788–3794, Hong Kong, 1-6 June 2008. IEEE Computational Intelligence Society, IEEE Press. doi: 10.1109/CEC.2008.4631311.

[169] Michael O'Neill and Anthony Brabazon. Recent patenets on genetic programming. *Recent Patents on Computer Science*, 2(1):43–49, 2009.

[170] Michael O'Neill and Conor Ryan. Automatic generation of programs with grammatical evolution. In Derek Bridge, Ruth Byrne, Barry O'Sullivan, Steve Prestwich, and Humphrey Sorensen, editors, *Artificial Intelligence and Cognitive Science AICS 1999*, number 10 in , University College Cork, Ireland, 1-3 September 1999.

[171] Michael O'Neill and Conor Ryan. Automatic generation of caching algorithms. In Kaisa Miettinen, Marko M. Mäkelä, Pekka Neittaanmäki, and Jacques Periaux, editors, *Evolutionary Algorithms in Engineering and Computer Science*, pages 127–134, Jyväskylä, Finland, 30 May - 3 June 1999. John Wiley & Sons.

[172] Michael O'Neill and Conor Ryan. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358, August 2001. ISSN 1089-778X. doi: 10.1109/4235.942529.

[173] Michael O'Neill and Conor Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language*, volume 4 of *Genetic programming.* Kluwer Academic Publishers, 2003. ISBN 1-4020-7444-1.

[174] Michael O'Neill and Conor Ryan. Grammatical evolution by grammatical evolution: The evolution of grammar and genetic code. In Maarten Keijzer, Una-May O'Reilly, Simon M. Lucas, Ernesto Costa, and Terence Soule, editors, *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, volume 3003 of *LNCS*, pages 138–149, Coimbra, Portugal, 5-7 April 2004. Springer-Verlag. ISBN 3-540-21346-5.

[175] Michael O'Neill, J. J. Collins, and Conor Ryan. Automatic generation of robot behaviours using grammatical evolution. In Masanori Sugisaka and Hiroshi Tanaka, editors, *Proceedings of the Fifth International Symposium on Artificial Life and Robotics*, pages 351–354, Oita, Japan, 26-28 January 2000.

[176] Michael O'Neill, Anthony Brabazon, Conor Ryan, and J. J. Collins. Evolving market index trading rules using grammatical evolution. In Egbert J. W. Boers, Stefano Cagnoni, Jens Gottlieb, Emma Hart, Pier Luca Lanzi, Gunther R. Raidl, Robert E. Smith, and Harald Tijink, editors, *Applications of Evolutionary Computing*, volume 2037 of *LNCS*, pages 343–352, Lake Como, Italy, 18-19 April 2001. Springer-Verlag. ISBN 3-540-41920-9.

[177] Michael O'Neill, Conor Ryan, Maarten Keijzer, and Mike Cattolico. Crossover in grammatical evolution. *Genetic Programming and Evolvable Machines*, 4(1):67–93, March 2003. ISSN 1389-2576. doi: 10.1023/A:1021877127167.

[178] Michael O'Neill, Anthony Brabazon, and Catherine Adley. The automatic generation of programs for classification problems with grammatical swarm. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 1, pages 104–110. IEEE, 2004.

[179] Michael O'Neill, Catherine Adley, and Anthony Brabazon. Grammatical evolution approach to eukaryotic promoter recognition. Poster Presentation, 16 February 2005.

[180] Michael O'Neill, Erik Hemberg, Conor Gilligan, Eliott Bartley, James McDermott, and Anthony Brabazon. GEVA: Grammatical evolution in java. *SIGEVOlution*, 3 (2), Summer 2008.

[181] Michael O'Neill, James McDermott, John Mark Swafford, Jonathan Byrne, Erik Hemberg, Anthony Brabazon, Elizabeth Shotton, Ciaran McNally, and Martin Hemberg. Evolutionary design using grammatical evolution and shape grammars: Designing a shelter. *International Journal of Design Engineering*, 3(1):4–24, 2010. doi: 10.1504/IJDE.2010.032820.

[182] Alfonso Ortega, Marina de la Cruz, and Manuel Alfonseca. Christiansen grammar evolution: Grammatical evolution with semantics. *IEEE Transactions on Evolutionary Computation*, 11(1):77–90, February 2007. ISSN 1089-778X. doi: 10.1109/TEVC.2006.880327.

[183] Michael ONeill, JJ Collins, and Conor Ryan. Automatic programming of robots. In *Proceedings of the 11th Irish conference on Artificial Intelligence and Cognitive Science (AICS2000)*, pages 23–25, 2000.

[184] Norman Paterson. *Genetic programming with context-sensitive grammars*. PhD thesis, Saint Andrew's University, September 2002.

[185] Judea Pearl. *Heuristics: intelligent search strategies for computer problem solving.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

[186] Diego Perez, Miguel Nicolau, Michael O'Neill, and Anthony Brabazon. Reactiveness and navigation in computer games: Different needs, different approaches. In *IEEE Conference on Computation Intelligence and Games, CIG 2011, Seoul, South Korea, August 31 - September 3, 2011, Proceedings*, pages 273–280. IEEE Press, 2011.

[187] Diego Perez, Miguel Nicolau, Michael O'Neill, and Anthony Brabazon. Evolving behaviour trees for the mario AI competition using grammatical evolution. In Cecilia Di Chio, Stefano Cagnoni, Carlos Cotta, Marc Ebner, Aniko Ekart, Anna I Esparcia-Alcazar, Juan J. Merelo, Ferrante Neri, Mike Preuss, Hendrik Richter, Julian Togelius, and Georgios N. Yannakakis, editors, *Applications of Evolutionary Computing, EvoApplications 2011: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, EvoSTOC*, volume 6624 of *LNCS*, pages 123–132, Turin, Italy, 27-29 April 2011. Springer Verlag. doi: 10.1007/978-3-642-20525-5_13.

[188] Michael Phelan and Seán McGarraghy. Mitigating the bullwhip effect in supply chains using grammatical evolution. In *Proceedings of the 25th International Conference of the System Dynamics Society*, Boston, 2007.

[189] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A field guide to genetic programming.* Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

[190] Petr Pospichal, Eoin Murphy, Michael O'Neill, Josef Schwarz, and Jiri Jaros. Acceleration of grammatical evolution using graphics processing units: computational intelligence on consumer games and graphics hardware. In Simon Harding, W. B. Langdon, Man Leung Wong, Garnett Wilson, and Tony Lewis, editors, *GECCO 2011 Computational intelligence on consumer games and graphics hardware (CIGPU)*, pages 431–438, Dublin, Ireland, 12-16 July 2011. ACM. doi: 10.1145/2001858.2002030.

[191] John Reddin, James McDermott, and Michael O'Neill. Elevated pitch: Automated grammatical evolution of short compositions. In Mario Giacobini, Ivanoe De Falco, and Marc Ebner, editors, *Applications of Evolutionary Computing, EvoWorkshops2009: EvoCOMNET, EvoENVIRONMENT, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoINTERACTION, EvoMUSART, EvoNUM, EvoPhD, EvoSTOC, EvoTRANSLOG*, volume 5484 of *LNCS*, pages 579–584, Tubingen, Germany, 15-17 April 2009. Springer Verlag. doi: 10.1007/978-3-642-01129-0_65.

[192] Torsten Reil. Dynamics of gene expression in an artificial genome - implications for biological and artificial ontogeny. In *Proceedings of the 5th European Conference on Artificial Life (ECAL99), number 1674 in Lecture Notes in Artificial Intelligence*, pages 457–466. Springer-Verlag, 1999.

[193] Joseph Reisinger, Kenneth O. Stanley, and Risto Miikkulainen. Towards an empirical measure of evolvability. In *Proceedings of the 2005 workshops on Genetic and evolutionary computation*, GECCO '05, pages 257–264, New York, NY, USA, 2005. ACM. doi: 10.1145/1102256.1102315.

[194] Elaine Rich. *Artificial intelligence*. McGraw-Hill, Inc., New York, NY, USA, 1983.

[195] Franz Rothlauf and Marie Oetzel. On the locality of grammatical evolution. In Pierre Collet, Marco Tomassini, Marc Ebner, Steven Gustafson, and Anikó Ekárt, editors, *Proceedings of the 9th European Conference on Genetic Programming*, volume 3905 of *Lecture Notes in Computer Science*, pages 320–330, Budapest, Hungary, 10 - 12 April 2006. Springer. ISBN 3-540-33143-3. doi: 10.1007/11729976_29.

[196] Conor Ryan and R. Muhammad Atif Azad. Sensible initialisation in chorus. In Conor Ryan, Terence Soule, Maarten Keijzer, Edward Tsang, Riccardo Poli, and Ernesto Costa, editors, *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 394–403, Essex, 14-16 April 2003. Springer-Verlag. ISBN 3-540-00971-X.

[197] Conor Ryan and Michael O'Neill. Grammatical evolution: A steady state approach. In John R. Koza, editor, *Late Breaking Papers at the Genetic Programming 1998 Conference*, pages 180–185, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Stanford University Bookstore.

[198] Conor Ryan, Atif Azad, Alan Sheahan, and Michael ONeill. No coercion and no prohibition, a position independent encoding scheme for evolutionary algorithms the chorus system. In JamesA. Foster, Evelyne Lutton, Julian Miller, Conor Ryan, and Andrea Tettamanzi, editors, *Genetic Programming*, volume 2278 of *Lecture Notes in Computer Science*, pages 131–141. Springer Berlin Heidelberg, 2002. ISBN 978-3-540-43378-1. doi: 10.1007/3-540-45984-7_13.

[199] Yves Schabes and Aravind K Joshi. Parsing with lexicalized tree adjoining grammar, 1990.

[200] Yves Schabes and Stuart M Shieber. An alternative conception of tree-adjoining derivation. *Computational Linguistics*, 20(1):91–124, 1994.

[201] Sevil Sen and John Andrew Clark. A grammatical evolution approach to intrusion detection on mobile ad hoc networks. In *WiSec '09: Proceedings of the second ACM conference on Wireless network security*, pages 95–102, Zurich, Switzerland, March 16-19 2009. ACM. doi: 10.1145/1514274.1514289.

[202] Y. Shan, H. Abbass, R. I. McKay, and D. Essam. AntTAG: a further study. In Ruhul Sarker and Bob McKay, editors, *Proceedings of the Sixth Australia-Japan Joint Workshop on Intelligent and Evolutionary Systems*, Australian National University, Canberra, Australia, 30 November 2002.

[203] Jianhua Shao, James McDermott, Michael O'Neill, and Anthony Brabazon. Jive: a generative, interactive, virtual, evolutionary music system. In Cecilia Di Chio, Anthony Brabazon, Gianni A. Di Caro, Marc Ebner, Muddassar Farooq, Andreas Fink, Jorn Grahl, Gary Greenfield, Penousal Machado, Michael O'Neill, Ernesto Tarantino, and Neil Urquhart, editors, *EvoMUSART*, volume 6025 of *LNCS*, pages 341–350, Istanbul, 7-9 April 2010. Springer. doi: 10.1007/978-3-642-12242-2_35.

[204] Hiroaki Shimooka and Yoshiji Fujimoto. Generating equations with genetic programming for control of a movable inverted pendulum. In R. I. Bob McKay, X. Yao, Charles S. Newton, J.-H. Kim, and T. Furuhashi, editors, *Simulated Evolution and Learning: Second Asia-Pacific Conference on Simulated Evolution and Learning, SEAL'98. Selected Papers*, volume 1585 of *LNAI*, pages 179–186, Australian Defence Force Academy, Canberra, Australia, 24-27 November 1998. Springer-Verlag. published in 1999.

[205] Hiroaki Shimooka and Yoshiji Fujimoto. Generating robust control equations with genetic programming for control of a rolling inverted pendulum. In Darrell Whitley, David Goldberg, Erick Cantu-Paz, Lee Spector, Ian Parmee, and Hans-Georg Beyer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 491–495, Las Vegas, Nevada, USA, 10-12 July 2000. Morgan Kaufmann. ISBN 1-55860-708-0.

[206] Jungseok Shin, Moonyoung Kang, Bob McKay, Xuan Nguyen, Tuan-Hao Hoang, Naoki Mori, and Daryl Essam. Analysing the regularity of genomes using compression and expression simplification. In Marc Ebner, Michael O'Neill, Anikó Ekárt, Leonardo Vanneschi, and Anna Isabel Esparcia-Alcázar, editors, *Proceedings of the 10th European Conference on Genetic Programming*, volume 4445 of *Lecture Notes in Computer Science*, pages 251–260, Valencia, Spain, 11-13 April 2007. Springer. ISBN 3-540-71602-5. doi: 10.1007/978-3-540-71605-1_23.

[207] Lee Spector and Kilian Stoffel. Ontogenetic programming. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 394–399, USA, 1996. MIT Press.

[208] Rainer Storn and Kenneth Price. Differential evolution a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997. ISSN 0925-5001. doi: 10.1023/A:1008202821328.

[209] John Swafford, Miguel Nicolau, Erik Hemberg, Michael O'Neill, and Anthony Brabazon. Comparing methods for module identification in grammatical evolution. In Terry Soule, Anne Auger, Jason Moore, David Pelta, Christine Solnon, Mike Preuss, Alan Dorin, Yew-Soon Ong, Christian Blum, Dario Landa Silva, Frank Neumann, Tina Yu, Aniko Ekart, Will Browne, Tim Kovacs, Man-Leung Wong, Clara Pizzuti, Jon Rowe, Tobias Friedrich, Giovanni Squillero, Nicolas Bredeche, Stephen Smith, Alison Motsinger-Reif, Jose Lozano, Martin Pelikan, Silja Meyer-Nienberg,

Christian Igel, Greg Hornby, Rene Doursat, Steve Gustafson, Gustavo Olague, Shin Yoo, John Clark, Gabriela Ochoa, Gisele Pappa, Fernando Lobo, Daniel Tauritz, Jurgen Branke, and Kalyanmoy Deb, editors, *GECCO '12: Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, pages 823–830, Philadelphia, Pennsylvania, USA, 7-11 July 2012. ACM. doi: 10.1145/2330163.2330277.

[210] John Mark Swafford. *Analyzing the Discovery and Use of Modules in Grammatical Evolution.* PhD thesis, University College Dublin, 2012.

[211] John Mark Swafford, Michael O'Neill, Miguel Nicolau, and Anthony Brabazon. Exploring grammatical modification with modules in grammatical evolution. In Sara Silva, James A. Foster, Miguel Nicolau, Mario Giacobini, and Penousal Machado, editors, *Proceedings of the 14th European Conference on Genetic Programming, EuroGP 2011*, volume 6621 of *LNCS*, pages 310–321, Turin, Italy, 27-29 April 2011. Springer Verlag. doi: 10.1007/978-3-642-20407-4_27.

[212] Kun Wang, Vinh Bui, and Hussein A. Abbass. Evolving stories: Tree adjoining grammar guided genetic programming for complex plot generation. In Kalyanmoy Deb, Arnab Bhattacharya, Nirupam Chakraborti, Partha Chakroborty, Swagatam Das, Joydeep Dutta, Santosh K. Gupta, Ashu Jain, Varun Aggarwal, Jürgen Branke, Sushil J. Louis, and Kay Chen Tan, editors, *Simulated Evolution and Learning - 8th International Conference, SEAL 2010, Kanpur, India, December 1-4, 2010. Proceedings*, volume 6457 of *Lecture Notes in Computer Science*, pages 135–145. Springer, 2010. doi: 10.1007/978-3-642-17298-4_14.

[213] James Watson, Janet Wiles, and Jim Hanan. Towards more relevant evolutionary models: Integrating an artificial genome with a developmental phenotype. In *In: Proceedings of the Australian Conference on Artificial Life (ACAL*, pages 288–298, 2003.

[214] David Jeremy Weir. *Characterizing mildly context-sensitive grammar formalisms.* PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1988.

[215] Peter Alexander Whigham. *Grammatical Bias for Evolutionary Learning.* PhD thesis, School of Computer Science, University College, University of New South Wales, Australian Defence Force Academy, Canberra, Australia, 1996.

[216] Darrell Whitley, Stephen Dominic, Rajarshi Das, and Charles W. Anderson. Genetic reinforcement learning for neurocontrol problems. *Machine Learning*, 13:259–284, 1993. ISSN 0885-6125.

[217] K. Willadsen and J. Wiles. Dynamics of gene expression in an artificial genome. In *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, volume 1, pages 185 – 190 Vol.1, dec. 2003. doi: 10.1109/CEC.2003.1299573.

[218] Dominic Wilson and Devindar Kaur. Using grammatical evolution for evolving intrusion detection rules. In *Proceedings of the 5th WSEAS International Conference on Information Security and Privacy*, ISP'06, pages 183–188, Stevens Point, Wisconsin, USA, 2006. World Scientific and Engineering Academy and Society (WSEAS). ISBN 960-8457-56-4.

[219] Man Leung Wong and Kwong Sak Leung. Evolutionary program induction directed by logic grammars. *Evol. Comput.*, 5(2):143–180, June 1997. ISSN 1063-6560. doi: 10.1162/evco.1997.5.2.143.

[220] Payam Zahadat and Kasper Stoy. An alternative representation of fractal gene regulatory networks facilitating analysis and interpretation. *Annals of Mathematics and Artificial Intelligence*, 65(4):285–316, 2012.

[221] Payam Zahadat, DavidJohan Christensen, UlrikPagh Schultz, Serajeddin Katebi, and Kasper Stoy. Fractal gene regulatory networks for robust locomotion control of modular robots. In Stphane Doncieux, Benot Girard, Agns Guillot, John Hallam, Jean-Arcady Meyer, and Jean-Baptiste Mouret, editors, *From Animals to Animats 11*, volume 6226 of *Lecture Notes in Computer Science*, pages 544–554. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-15192-7. doi: 10.1007/978-3-642-15193-4_51.

[222] Song Zhan, J.F. Miller, and A.M. Tyrrell. An evolutionary system using development and artificial genetic regulatory networks. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 815–822, 2008. doi: 10.1109/CEC.2008.4630890.

[223] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm, 2001.