A Flexible Representation for Genetic Programming: Lessons from Natural Language Processing

A thesis submitted to the

School of Information Technology and Electrical Engineering

University College

University of New South Wales

Australian Defence Force Academy

for the degree of Doctor of Philosophy

By

Nguyen Xuan Hoai

December 2004

Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no material previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgment is made in the thesis. Any contribution made to the research by colleagues, with whom I have worked at UNSW or elsewhere, during my candidature, is fully acknowledged.

I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.

Nguyen Xuan Hoai

Abstract

This thesis principally addresses some problems in genetic programming (GP) and grammar-guided genetic programming (GGGP) arising from the lack of operators able to make small and bounded changes on both genotype and phenotype space. It proposes a new and flexible representation for genetic programming, using a state-of-the-art formalism from natural language processing, Tree Adjoining Grammars (TAGs). It demonstrates that the new TAG-based representation possesses two important properties: non-fixed arity and locality. The former facilitates the design of new operators, including some which are bio-inspired, and others able to make small and bounded changes. The latter ensures that bounded changes in genotype space are reflected in bounded changes in phenotype space.

With these two properties, the thesis shows how some well-known difficulties in standard GP and GGGP tree-based representations can be solved in the new representation. These difficulties have been previously attributed to the tree-based nature of the representations; since TAG representation is also tree-based, it has enabled a more precise delineation of the causes of the difficulties.

Building on the new representation, a new grammar guided GP system known as TAG3P has been developed, and shown to be competitive with other GP and GGGP systems.

A new schema theorem, explaining the behaviour of TAG3P on syntactically constrained domains, is derived.

Finally, the thesis proposes a new method for understanding performance differences between GP representations requiring different ways to bound the search space, eliminating the effects of the bounds through multi-objective approaches.

Acknowledgments

The first person I would like to thank is my principal supervisor, Dr Robert (Bob) Ian McKay for introducing me to the field of genetic programming. Bob has been the ideal mentor that I was looking for. He taught me how to love research and how not to be a research-alcoholic. His genius has been a constant source of help. His encouragement and constructive criticism have been the triggers for much of the research work in this thesis.

I wish also to thank my co-supervisors Dr Daryl Essam and Dr Hussein Aly Abbass. It is a pleasure and luxury for me to work with such briliant people during my candidature. The seemingly never-ending, but always useful, discussions between us on the matters related to the work presented in this thesis will be in my mind forever.

I would like to express my gratitude to Professor Jason Daida for sending some of the figures in his papers on the problem of structural difficulty in GP and for also allowing me to reproduce them in chapter 8 of this thesis. I must also thank Dr Zitzler for promptly replying to my questions related to the implementation of his SPEA2 algorithm.

During my candidature, I have had opportunities to attend a number of top academic conferences in the field, such as: CEC, EuroGP and GECCO. I wish to thank the school of IT&EE, University of New South Wales at the Australian Defence Force Academy, for providing me the necessary funding to go and present my research papers at those conferences. While attending those conferences, I was lucky to have had a number of useful and interesting discussions with a number of researchers in the field. The exchange of ideas and information with them was very important to my research. For that reason, I wish to thank those people who were generous to me by giving information, suggestions, and discussions related to my work and research interests. I would like to mention some of their names in no particular order: Bill Langdon, Michael O'Neil, Man Leung Wong, Una-OReilly, Peter Whigham, Vic Ciesielski, Ivan Tanev, Karl Lehre, Sung-Bae Cho, Andras Kornai. Also, when I asked for people who have done some work related to grammar-guided genetic programming over the GP mailing list, a number of respondents stood out and sent me information about their work. Thank you.

During my period of three and a half years at ADFA, I shared my room with Shan Yin, a nice and briliant Chinese PhD student, whom I share interest in research, morden chinese history and politics. I wish to thank him for sharing with me many interesting discussions both related and not related to research. Hoang Tuan Hao, my former student, must also be thanked for helping me to draw some of the figures in this thesis.

I am also indebted to Professor Nguyen Xuan Huy at the Vietnamese Institution of Information Technology and Professor Pham The Long, Vice Rector of Vietnamese Military Technical Academy for their care, support, and encouragement when I was a university student in Vietnam.

Last, but most important, I would like to dedicate this work to my family, my wife, Nguyen Thi Thu Huyen for having gone side by side with me through all the ups and downs in life for the last five years, and to my parents, Nguyen Tai and Nguyen Thi Lai, for working so hard to bring up their son and for never losing the hope, that one day, he will become one of the best in his generation.

Contents

List of Figures

Chapter 1

Introduction

If you want to go far in life - make a series of small steps

Old Vietnamese Proverb

1.1 Motivation

Genetic programming (GP), a new paradigm of evolutionary algorithms, has attracted increasing interest from academic researchers and/or practitioners over the last decade. Consequently, GP has been widely developed, and applied to solve many real-world problems. One of the important extensions is grammar-guided genetic programming (GGGP), where formal grammars (usually string rewriting systems) are used to relax the closure requirement, which states that every primitive in GP must be of the same type, and set a chosen declarative bias on the space of the programs. Moreover, formal grammars can be used to set declarative bias on the search process of genetic programming.

Given GP's deep roots in Genetic Algorithm (GA) research, it is not surprising that many researchers and practitioners view GP as a natural extension of GA, simply extending GA's fixed-shape and -size chromosomal representation to tree-based structures with variable shape and size. However there is a price to pay. The natural topology on tree-based GP search spaces, corresponding to the Hamming distance in binary GA, is some form of edit distance. The tree structure in standard GP is of fixed arity, i.e a function of arity n must have exactly n children, wherever it occurs in the tree. This renders it very difficult to design operators which make small and bounded changes with respect to the topology, or to build natural analogues of many of the important biological variation operators. Operators making small changes are important both for theoretical purposes – helping to characterize the neighbourhood structure of the search space – and practically, as valuable operators in particular types of problems. Furthermore, biologically-based operators have formed an important source of inspiration for GAs. Arguably, their availability in GP would be a corresponding asset. These issues become even more acute when we move to GGGP (despite its many other advantages and successes) because the tree structure is even further constrained by the production rules of the grammar.

Viewing the tree-based nature of the representation as the culprit for these limitations, several researchers have developed linearisations of the tree representation in GP and in GGGP, introducing a genotype-phenotype mapping, with a linear genotype. This linearity means that these representations inherit some of the characteristics of the GA linear chromosome that have been lost in GP and GGGP tree-based representation. However, the genotype-phenotype mapping in these linearised GP and GGGP systems usually lack the locality (or causality) property, which states that small changes in the genotype of an individual cause small changes on the phenotype level. Thus even though there exist operators which can make small and bounded changes of the genotype, the corresponding effects on the phenotype may be unbounded and uncontrollable.

In the field of natural language processing, which has had very limited interaction with the GP and GGGP fields, a new formalism has become increasingly important to linguists: the tree adjoining grammar (TAG). TAGs are tree-rewriting systems proposed with linguistic motivations. An important observation from linguistics and natural language processing is that complex sentences can be built up from very simple ones by operations that act directly on the syntactic tree structure of sentences, modifying them bit by bit. The applications of the modification operations can themselves be recorded in a tree structure, recording the history of the transformation from the original simple sentence to a more complex one. This results in the concept of a TAG-derivation trees. One of the important properties of the TAG derivation tree is that a node can have a variable number of children. Each node records a particular modification operation which took place on the parent node; hence its presence or absence does not affect its siblings. In this thesis, we denote this property as non-fixed arity. It provides greater freedom to shuffle nodes or subtrees within the tree than is possible in fixed arity trees. Furthermore, any small change on the TAG-derivation tree (modification operation history tree) results in only a small change in the tree derived from it, since the modification operation defined in TAGs, adjunction, does not change the contents of the nodes that are already in the modified sentence tree, but adds a bounded number of new nodes, the bound being determined by the specific grammar. It is interesting whether some of the properties of TAG-derivation trees, which have been shown to be useful in the context of linguistics and natural language processing, are also useful in the field of GP and GGGP, and in particular, whether they can help to solve some of the problems mentioned above. This thesis is an attempt to provide an answer to that question.

1.2 Statement of the Thesis

This thesis describes a number of representation-related issues arising from the extension from genetic algorithms (GAs) to genetic programming (GP), and to grammar guided genetic programming (GGGP). We argue that these difficulties are the consequence of the lack of operators that can make small and bounded changes.

A new GP representation, called TAG-based representation, is proposed, in which the structure of programs is also a tree. However, the new tree-based representation is non-fixed arity rather than fixed arity as in GP representation, and it is object-based rather than rule-based as in GGGP representation. The thesis shows how a wide variety of bio-inspired and other operators can be supported in TAG-based GP because of the non-fixed arity property, and also how a number of important related difficulties in tree-based representation can be solved.

The thesis proposes a new grammar guided genetic programming system, based on TAG representation and known as TAG3P. It is shown to be competitive with other standard GP and GGGP systems. A schema structure for TAG-based representation is defined, which unifies the three principal aspects of schemata on syntactically constrained domains. A simple schema theorem for TAG3P is derived, helping to cast light on how it works.

Since TAG-based representation naturally uses size as a search-space bound, whereas the natural comparators, GP and GGGP, most naturally use depth, the thesis must necessarily face the issue of how to compare algorithms with different search spaces; and in particular, how to determine whether differences in performance are the result of differences in representation, operators and fitness landscape, or merely the result of different search space size. The thesis proposes an alternative comparison method, using multi-objective search, eliminating the effects of search space bounds by eliminating the bounds themselves.

1.3 Outline of Dissertation

Chapter 2 give a brief overview of related work in genetic programming, and a survey of grammar guided genetic programming. Some issues in the current GP and GGGP representations are formulated. The chapter is intended to provide a general understanding of the history in the development of grammar-based genetic programming systems.

The body of the thesis is intended to cover a wide spectrum of topics and issues in the comparison between GP systems (and in particular, TAG3P); these are not particularly closely connected to each other except through their relevance to the TAG3P comparisons. Rather than present a somewhat disconnected series of backgrounds on particular issues in this chapter, we decided to include an overview of the background and related work on each particular topic at the beginning of the relevant chapter (chapters 6,7,8,9,10).

The concepts of tree adjoining grammars (TAG) and their derivation trees are given in chapter 3. In that chapter, a new form of TAG derivation tree, combining important aspects of two previous formulations, is given. This forms the basis for a new GP representation. The chapter serves as the foundation for the thesis, giving a framework and arguments for the new representation.

The rest of the thesis, except the last chapter, can be divided into two parts. Part 1 includes chapters 4, 5 and 6, and a part of chapter 7. This part describes the TAG3P system, built upon the representation in chapter 3. In detail, chapter 4 presents the design of TAG3P. Chapter 5 gives some examples, presenting some experimental results for TAG3P on some standard problems from the GP literature, and comparing the results with standard GP and GGGP systems. Based on some unique features of the representation, some further analyses are also presented to probe deeper into the TAG3P evolutionary process than is possible with the other representations. Chapter 6 presents some new operators made possible by the non-fixed arity property, and demonstrates their value on specific problems. Chapter 7 provides a theoretical framework for TAG3P by formulating a schema concept and proving an inexact schema theorem.

Part 2 includes the rest of chapter 7, together with chapters 8, 9 and 10. This part shows how TAG-based representation can provide solutions to some difficulties in GP and GGGP. In chapter 7, we show how it is possible to define rooted schemata for a syntactically constrained domain with TAG3P because of the object-based aspect of the representation. We explain how this schema definition unifies three important aspects of schemata on syntactically constrained domains, as component of the representation, as search subspace, and as a sub-language of the grammar. Chapter 8 provides a hypothesis for the cause of a well-know problem in GP, Daida's problem of structural difficulty. Using TAG-based representation, and in particular its structural mutation operators, we show that the problem can be greatly ameliorated, providing evidence for our hypothesis as to the cause of the problem. Chapter 9 presents what is, to the best of our knowledge, the first fitness landscape analysis on syntactically constrained domains. This is facilitated by the TAG representation's facilitation of the design of bounded-change operators, and the locality (causality) property. In particular, that chapter studies the effects of changing target functions over a family of functions, and of changing the language bias (grammar) on the difficulty of the fitness landscape. Chapter 10 presents an alternative approach to comparing GP representations that use different methods of bounding search space size. In this approach, the use of multi-objective selection pressure enables us to separate the effects on performance resulting from different bounds on the complexity of individual programs from the more interesting effects arising from differences in representation and operators.

Chapter 11 summarises the thesis, reviews the contributions, and comments on future directions. Five Appendices are provided, incorporating some more detailed background material and some detailed experimental results. A bibliography is presented at the end of the thesis.

Chapter 2

Related Work

In this chapter, we survey previous research on representation for genetic programming and grammar guided genetic programming. We also explore and discuss some problems which arise in the extension of ideas from genetic algorithms to genetic programming (GP), and to grammar-guided genetic programming (GGGP).

2.1 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are a class of search algorithms that simulate biological evolution [Bac1996]. The simulation can be seen in three aspects. An EA uses a population of individuals; those individuals compete with each other, according to Darwin's principle of natural selection, to reproduce; and the evolution is carried out by random variation in the genetics of chosen individuals.

Since the pioneering work by Friedberg ([Fri1958, Fri1959]),Fogel ([Fog1966]), Schwefel ([Shw1968]) Rechenberg[Rec1973]), and Holland ([Hol1975]), EAs have diverged into three main streams, namely evolution strategies, evolutionary programming and genetic algorithms. However, the basic algorithms of all EAs are still very similar, as shown in the flowchart in Figure 2.1.

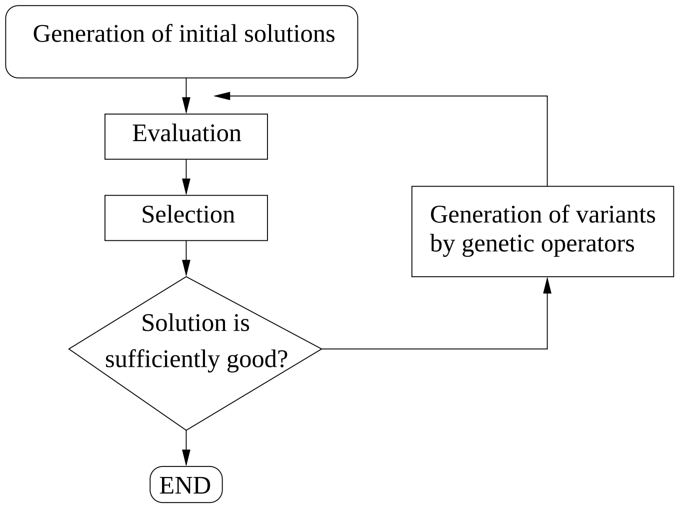

Generation of initial solutions
Evaluation
Selection
Generation of variants by genetic operators
Solution is sufficiently good?
END

Figure 2.1: The basic flow chart for evolutionary algorithms (EAs) [Fog1995]

2.2 The Genetic Algorithm

The original genetic algorithm (GA) proposed by Holland ([Hol1975]) is currently the most popular evolutionary algorithm. It is known as the standard (or simple) GA. It has been intensively studied, and shown to be useful in solving a wide range of real-world problems ([Mit1996]). The GA is viewed as the predecessor of genetic programming (GP), from which the latter gets its name.

The original GA uses linear and fixed length binary strings as the representation for its chromosomes as depicted in Figure 2.2.

In [Hol1975] three genetic operators were proposed, namely, one-point crossover,

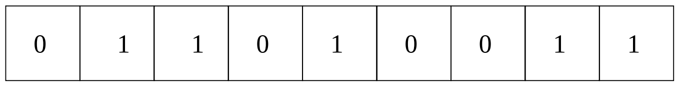


Figure 2.2: Representation for individuals in standard GA

one-point mutation, and inversion. Crossover was designated as the main genetic operator. It resembles genetic recombination in biological genome evolution. The one-point crossover between two chromosomes is done by exchanging segments of strings. One-point mutation and inversion are secondary mutation operators, which are used to keep a degree of genetic diversity in the population. One-point mutation is implemented as the flipping of a random bit in the string, and inversion is the rewriting a segment of the string in reverse order. Figure 2.3 shows how one-point crossover, one-point mutation, and inversion work.

Since Holland invented the first GA, there have been a wide variety of new GA

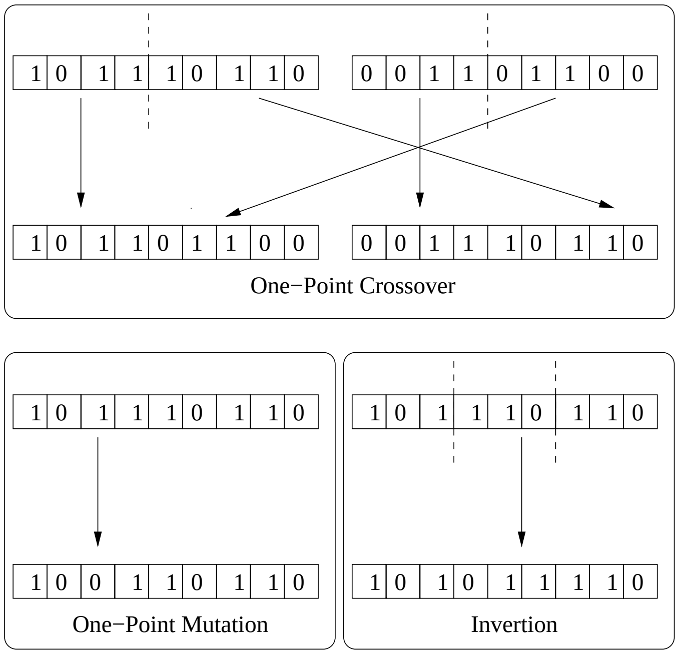


Figure 2.3: Basic genetic operators in standard GA

systems using different representations such as real coding, gray coding, messy coding, variant length chromosomes, and so on [Mit1996]. It is important to note that all these codings use a linear representation of the chromosome. The linearity of chromosomes facilitates the design of many different genetic operators. Indeed, from three operators in the beginning, there have been a vast number of new and bio-inspired ones derived in the literature [Mit1996, Bac2000a, Bac2000b]. Moreover, it is not difficult in GA's linear chromosome representation to design operators that make limited changes. In other words, if there is a topological structure (neighbourhood) on GA's linear chromosome, it is easy to design operators that respect this structure. For example, in the standard GA linear binary representation, if we use Hamming distance to define the neighbourhood structure on GA chromosome space, the one-point mutation operator makes minimal change in terms of Hamming distance (1), and therefore respects the topological order defined by Hamming distance, i.e it transforms a point to another point in its neighbourhood.

2.3 Genetic Programming

Genetic programming has been defined as a machine learning method to evolve computer programs by evolutionary means [Ban1998]. It is inspired by GAs, and seen as a branch of GAs by many researchers. The distinction between the two is partly philosophical, and there are generally exceptions for any nominated distinctions. However, it is generally agreed that their task and motivation are different. While GAs are commonly used for optimisation task, GP systems are usually used for learning. The scenario that a GA might be used to optimise a function given the definition of the function, while GP might be used to learn the definition of the function itself from sampled data, is an almost perfect illustration of the distinction between GAs and GP. The more important point for distinguishing between GA and GP systems is that GP systems usually employ more flexible chromosome representations, with variable size and shape. Therefore, it is believed that GAs are suited to the task of optimising parameters for solutions when their structure is known, while GP is preferred if the task is to learn and discover both content and structure of solutions [Ban1998]. GP has been successfully used to solve a vast ranges of real-world problems ([Ban1998] - chapter 12).

The early dawn of program evolution (or in Koza's words "the programming of computer by means of natural selection") includes work by Friedberg ([Fri1958, Fri1959]) on a learning machine, where the task is to teach a computers to learn to program for itself. Fogel et al. ([Fog1966]) proposed the evolutionary programming method to learn finite state machines (FSM), which are simple form of computer programs. Smith ([Smi1980]) in 1980 was probably the first researcher to study a variant of classifier systems, in which the individuals have variable sizes. Cramer introduced more features resembling the GP we know today, in particular tree representation and subtree crossover ([Cra1985]). In 1987, Dickmanns, Schmidhuber, and Winklehofer publish a paper, in which, the authors presented how to evolve programs with loops using genetic algorithms [Dic1987]. In his diploma thesis, Schmidhuber even went further on developing meta learning for program evolution. His system that can use the techniques in program evolution to learn for program evolution itself (learning to learn) [Sch1987].

2.3.1 Standard tree-based GP

Althought there had been a number of work that was very closed to genetic programming, the field of genetic programming was first clearly delineated by Koza in his seminal book [Koz92], which first demonstrated the power of GP and set the foundations for the field. In his work, a program induction system based on Darwin's natural selection principles was proposed. His system, which in this thesis is called standard GP, consists of five components: a representation for programs, a procedure for initialising the population of programs, a fitness evaluation procedure, genetic operators, and parameters.

Program Representation

Program representation in standard GP is Lisp-expression trees (or in brief, expression tree in this thesis) where the nodes are labeled by primitives from the set of function (F) and terminal (T) symbols. The function set $F = (f_1, f_2, ..f_n)$ consists of functions with arity (number of children or arguments) greater than 0, whereas the terminal set $T = (t_1, t_2, ..., t_m)$ contains 0-arity functions or constants. We note that all primitives ($F \cup T$) in standard GP have fixed arities. In other words, if a primitive h is of t-arity, it must have exactly t children in any individual. Figure 2.4 shows the program representation for standard GP.

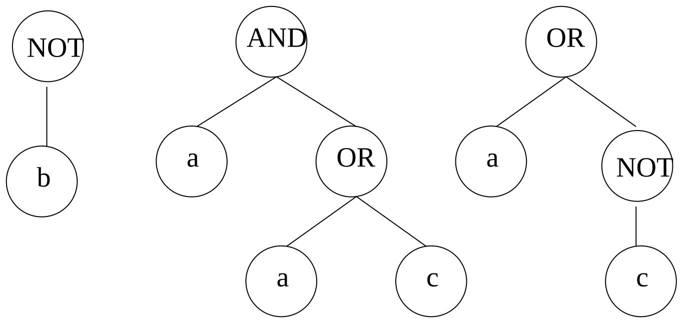


Figure 2.4: Lisp-expression tree representation in standard GP, where $F = (AND, OR, NOT)$ and $T = (a, b, c)$

Creating Initial Population

There are two procedures in standard GP for creating an initial population of programs, namely ramped and ramped-half-and-half. Ramped initialisation of a program (expression tree) commences by selecting a function f_i from F at random. For each argument p_i of f, this process is repeated with either a function or terminal being randomly picked to fill each argument position. If a terminal is selected, this branch of the function is terminated. If a function is chosen, the process is recursively called for that function. A maximal expression tree depth is usually used to limit the size of the initial program. Ramped initialisation of the population uses ramping for all individual programs. Ramped-half-and-half initilisation is the application of ramped initialisation for half of the population with, for the other half, full expression trees being generated randomly. The depth of the full trees is the specified maximal depth. Each of them is generated by a procedure that is similar to ramped initialisation of a program. However, before reaching the maximal depth, only primitives in F are used to grow the tree, whereas upon reaching the maximal depth, only primitives from T are selected.

Fitness Evaluation

Each program is assigned a numerical value called fitness. This fitness is obtained by evaluating the program against a set of test problems called fitness cases. These fitness cases represent the environment that the program is attempting to learn. The fitness of each program is usually normalised before selection (described below) is applied. The process for normalising expression tree programs in standard GP can be found in [Koz92].

Genetic Operators

The operators used in standard GP are a selection mechanism, reproduction, crossover, and mutation. Selection mechanisms are the way individual programs are selected from the population to enter the mating pool, where the genetic recombination and other operations take place. There are a substantial number of selection mechanisms in evolutionary algorithms, all of which are usable in standard GP [Bac1997] (part C2). Among them, the most popular are fitness-proportionate selection and tournament selection. In fitness-proportionate selection (otherwise known as roulette wheel selection), each program i is selected with probability $p_i = \frac{f_i}{\sum_{j=1}^{n} f_j}$, where f_i is the fitness of the individual i. In tournament selection, m individuals are randomly selected from the population to go though a competition, and the winner (in terms of fitness) is selected to go to the mating pool. Reproduction selects a program using the selection mechanism, and copies it to the next generation. Crossover (subtree crossover) selects two parents using the selection mechanism, and creates two children by swapping subtrees between the two parent programs. Figure 2.5 depicts how crossover works. In a mutation (subtree mutation) of an expression tree, a node is selected at random and the subtree below this node is deleted. A new subtree is randomly generated to replace the deleted subtree. Figure 2.6 shows how mutation works. In [Koz92], it is suggested that crossover and reproduction should be the main operators for GP, while the use of mutation should be very limited.

Parameters

Parameters in standard GP include the population size (POPSIZE), the maximal number of generations (MAXGEN), the maximal depth for initialisation and

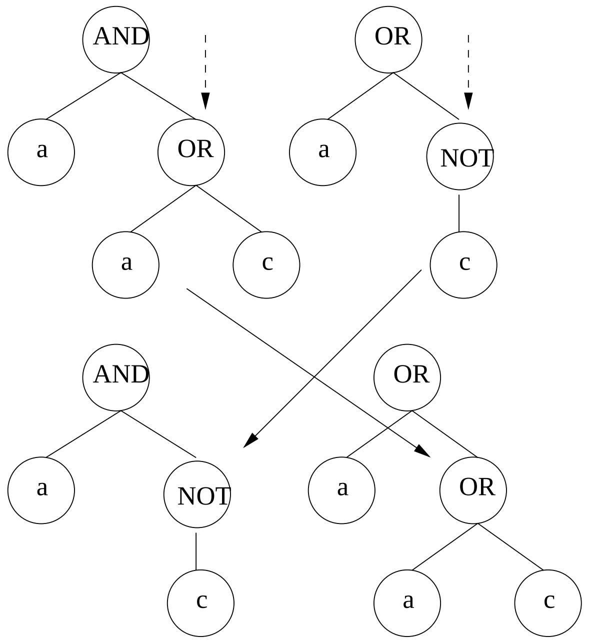


Figure 2.5: Crossover operator in standard GP

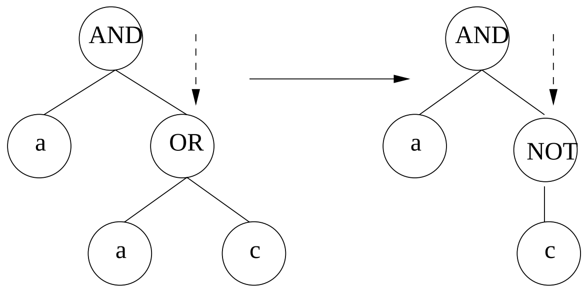


Figure 2.6: Mutation operator in standard GP

for the whole evolutionary process (MAXDEPTH), and the probabilities for the application of genetic operators (operator rates). Recently, GP has been extended to include ADF (automatically-defined functions) [Koz2004]. Consequently, a number of other parameters have been added such as: number of ADFs, number of each ADF's parameters, probability for ADF architecture altering operators, etc [Koz2004].

Sufficiency and Closure Requirements

In [Koz92], Koza stated two conditions that a GP system must satisfy in order to be applicable to a specific problem, namely, sufficiency and closure. Sufficiency requires GP to have a sufficient primitive set (functions and terminals) to represent the problem solutions [Koz92]. The closure condition was expressed in [Koz92] as a requirement that all functions and terminals must be of the same type, so that the application of genetic operators does not result in invalid expression trees. Therefore, the problem domain of standard GP is restricted to typeless problems. This limitation was one of the motivations for grammar guided genetic programming (described in the next section).

2.3.2 Some Problems with Tree-Based Genetic Programming

Viewed as a descendant of GAs, standard GP has inherited many aspects of GAs such as the evolutionary process, heavy use of crossover, and so on. It is tempting to believe that because of this ancestry, many properties and concepts from GA will transfer naturally to GP.

However, this is not necessarily the case. The representation for the chromosome in standard GP is a fixed arity expression tree. Compared with linear (and usually fixed size) GA chromosomes, it has much greater complexity. While in GA the evolution only affects the contents of genes in the chromosome, in standard tree-based GP the evolution proceeds in three dimensions: content, shape, and size. With the tree-based structure, it is not obvious that useful bio-inspired operators such as insertion, deletion, duplication, relocation, etc can be implemented in GP in the way they were on GA chromosomes (though [Koz92] proposed a permutation operator which was claimed to resemble inversion in GAs). Moreover, unlike GAs, it is not clear what might be a natural topological structure on the space of tree-based genotypes in GP [Lan2002], or that one can design operators that respect such topological structures (i.e operators which only make bounded changes, so that they can be used to explore the neighbourhood structure. Lacking a natural topological structure, and operators which make bounded changes within the topology, means it is more difficult with GP than

GAs to study various aspects involving the concepts of neighbourhood and adjacency, including fitness landscape investigation, application of heuristic search strategies that require neighbourhood definitions, etc.

The crossover and mutation operators in standard GP are indeed very disruptive. Since the early days, there have been a number of studies suggesting that crossover in standard tree-based GP is too disruptive [Nor1995a, Nor1996]. Some researchers went further, suggesting that crossover in standard GP acts merely as a macro-mutation operator ([Lan1995]). However it is in mutation that we can see the greatest incompatibility between GAs and standard GP. In the simple GA, the one-point mutation operator resembles natural mutation in biological genome evolution, where mutation usually involves a very small change made on the genotype [Rid1996]. Subtree mutation in standard GP is very disruptive, and the degree of change caused by it is not bounded or predictable. If the subtree mutation site is a leaf of the expression tree, and the new subtree generated by subtree mutation is small, then the degree of change is small. However, if the site of subtree mutation is at the root of the expression tree, it could change the tree completely.

We note that although the problems in this subsection are only discussed for tree-based representation, they appear to be equally applicable to GP systems with more complicated program representations, such as graph-based GP ([Tel1996], [Pol1999]).

2.3.3 GP with Linear Representation

The problems mentioned in the previous subsection could melt away with a linear representation for programs. A number of linear GP representations have been proposed in the literature. They can be divided into three groups: stack-based GP, machine-code GP (otherwise known as linear GP) and gene expression programming (GEP).

In stack-based GP [Per1994, Kei1994, Spe2001], each program instruction takes its arguments from a stack. After performing the calculation, it pushes the result back onto the stack. To guard against the problems of stack overflow and underflow, some checks are needed. When the checks detect these stack defects, it might allow the program to continue its execution by supplying default values on underflow and discarding data on overflow.

In machine-code GP (linear GP) ([Nor1997, Ban1998]), data are stored in registers. Each instruction reads its data from one register and write its output to another. The program inputs are written to registers before the program is executed, and its output is given by the final contents of one ore more registers. The instruction are direct binary machine codes, making the linear GP evaluation much faster than other GP systems.

Of the extant non-grammar based GP systems, possibly the closest to the approach in this thesis, especially in its emphasis on representation and operators, is gene expression programming (GEP) ([Fer2001], [Fer2002a], [Fer2002b], [Fer2002c], and [Fer2002d]). In GEP, a genotype to phenotype mapping is used. The phenotype is a standard GP expression tree, and the genotype is represented as follows. First, all the primitives (functions and terminals) used to solve problems in standard GP are indexed by integer numbers. The genotype is a linear and fixed length string of integers taken from $\{0..n\}$ (n is the cardinality of the primitive set). It is divided into two parts. The first and left part is filled with integers representing either function or terminal symbols. The second and right part is filled with integers representing terminal symbols only. The length of the first part is l, while the length of the second part is $h \times l$, making the total (fixed) length of the genotype $(h + 1) \times l$ – where l is a predefined integer (parameter) and h is the maximal arity over all the primitives. The translation from genotype to phenotype, which is a standard GP expression tree, is processed from left to right. At each step, a primitive (function or terminal) represented by the current gene is picked to fill the next uncompleted branch of the phenotype in top-down and left-to-right fashion. The process is terminated if the expression tree built from the genotype is completed. If there are still unused genes when the phenotype is completed, they are ignored and considered as introns (the part of the genotype that is not coding for the phenotype). At the completion of processing of the left part of the genotype, there can be at most l function symbols with unfilled arguments (in practice, rather less), and each has at most h arguments, so that there can be at most $h \times l$ unfilled arguments when processing of the right part is initiated. Since the right part is filled with terminals only, and the its length is $h \times l$, it is not possible to run out of genes in the translation from a genotype to its phenotype (thus avoiding a counterpart to the stack underflow problem of stack GP). Therefore for every valid genotype, a valid phenotype is created by the translation process. Figure 2.7 depicts the genotype structure and an example of translation from genotype to phenotype in GEP.

As claimed in [Fer2001, Fer2002a] the linearity of the GEP reprentation is one of

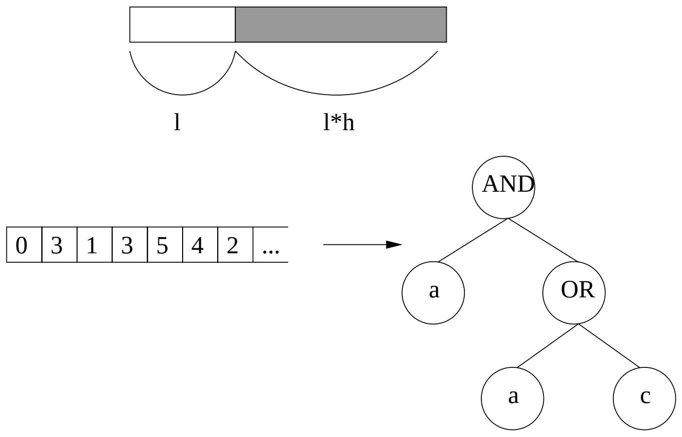


Figure 2.7: Genotype and Phenotype in GEP, where AND=0, OR=1, NOT=2, a=3, b=4, c=5.

its main advantages over standard GP. It allows GEP to design and experiment with various bio-inspired operators in order to study how they can be useful in the context of genetic programming. They also capable of making bounded and small change on the genotype level just as in GAs.

However, one problem with GEP linear representation (and we believe it is equally applicable to stack GP, and to some degree to linear GP) is that its mapping from the genotype to the phenotype space violates the causality principle for representations for evolutionary algorithms [Rec1973, Pal1994b, Sen1997], which states that "small changes in genotype should result in small changes in phenotype". In other words, the causality principle states that if a designer wants to obtain genetic operators with locality property (i.e, making small and bounded changes), it is essential for him to ensure the mapping between genotype and phenotype space causal. In GEP case, Although the change on the genotype caused by an operator can be bounded and small, the effect on the phenotype is hard to control. In particular, even a small change in genotype can change vastly the expressiveness (coding or non-coding) of the genes after the position where the change takes place. For example, if the first gene of the genotype is mutated, the phenotype could be changed completely. In other words, although the operator respects the topological structure of the genotype space, this effect is not replicated in the phenotype space. Figure 2.8 show an example of this situation.

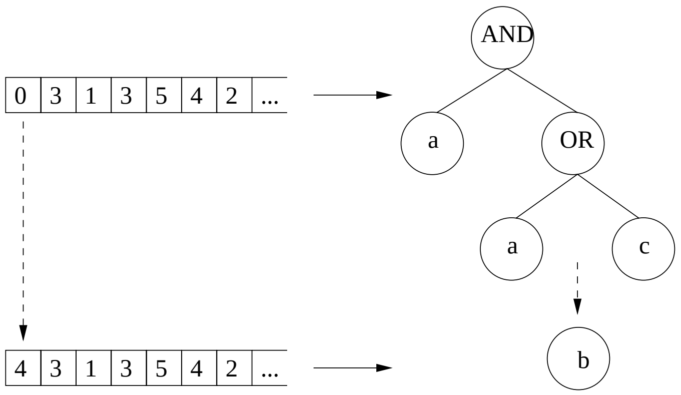


Figure 2.8: Genotype to phenotype map in GEP is not causal. The top-left corner is an genotype and the top-right corner is its corresponding phenotype. Just by one mutation taken in the first gene (bottom-left corner), the phenotype has changed completely (bottom-right corner)

2.4 Grammar Guided Genetic Programming

As mentioned earlier, the closure requirement restricts the domain of problems solved by standard GP to typeless problems. However, in reality, many problem domains are typed. For example many problems in decision tree induction in data mining ([Fre2002]), require that attributes have different types. Consequently, the space of expression trees used in standard GP would contain a vast number of infeasible individuals (i.e their fitness is not calculable). Even were it to start with a population of entirely valid individuals, the genetic operators might transform them into invalid ones during the evolutionary process.

One solution is to constrain the primitives of GP (both functions and terminals) by attaching types to them. The genetic operators are designed so that they respect the types of the primitives. This results in a branch of genetic programming, called strongly-typed genetic programming (STGP), proposed by Montana ([Mon1994, Mon1995]). Figure 2.9 depicts an example of representation and crossover used in strongly-typed GP.

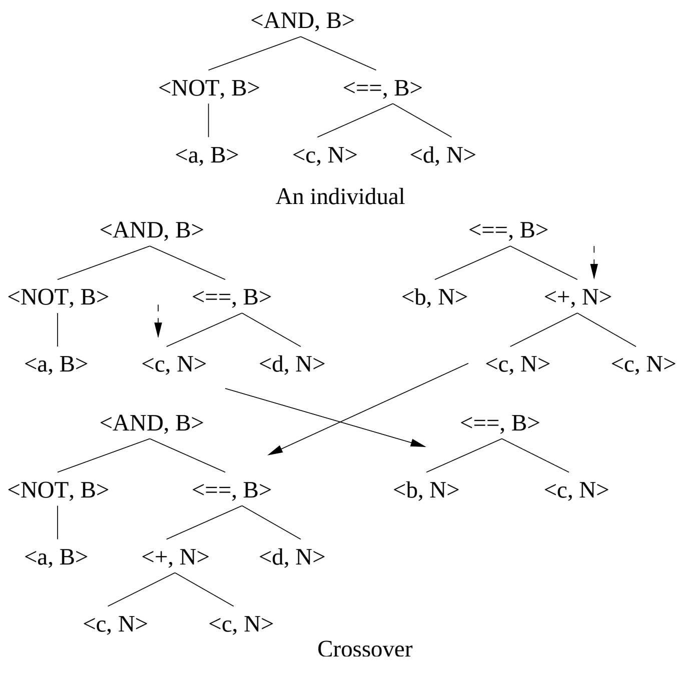


Figure 2.9: Strongly Typed GP

One important aim of STGP is to reduce the search space by avoiding the exploration of invalid programs. Therefore, it increases the chances of finding solutions compared with typeless GP systems. For instance, in [Hay1995b], it was demonstrated that by searching in the space of valid programs only, STGP outperformed standard GP for the problem of evolving cooperation strategies. STGP has been found useful in solving problem in various typed domains ([Hay1995a], [Hay1996b], [Har1997] [Sar1999],[Alb1999]). Extensions to STGP have been made by the use of a type hierarchy and type inheritance in the manner of object-oriented programming [Hay1996b, Bru1996]. By doing so, it helps to extend the constraints to multiple levels of nodes in tree-based programs, rather than just one level between parent and child nodes as in the original STGP. STGP has become highly popular, as almost all modern implementations of GP software provide users with the capability of using strong typing.

Another approach to relax the closure requirement in genetic programming is to use formal grammars to syntactically constrain the programs. Outside GP, formal grammars have been widely used to defined languages for a number of typed domains, such as programming and natural languages [Mol1988]. There has been a substantial volume of work in using grammar-based constraints on GP programs. They have shaped a subfield of GP, which we call grammar guided genetic programming in this thesis (GGGP). A number of definitions and related concepts for the different types of grammars mentioned in this section are given in Appendix A of the thesis.

We note that overcoming the restrictions of the closure requirement in GP is not the only motivation for using formal grammars. There are a number of reasons suggested in the literature of GGGP. Firstly, the use of grammars helps to encode the domain knowledge about the syntactical structure of programs. Since the domain knowledge might constrain the structure of solutions (such as that it must start with a certain function), the use of grammars helps to capture this structure by restricting the corresponding formal languages. In [Koz92] (chapter 19), some simple and ad hoc mechanisms were used to encode the known syntactical structures of programs in standard GP. However the use of formal grammars provides a more flexible and general way to deal with the issue. Secondly, by using grammars, it is easy to re-bias the syntactical structure of programs by changing grammars. Since the grammar is external to the GP system, rather than being a component, the bias can be tuned by changing the grammar without reimplementing the whole system to suit the bias. Since the grammar is declarative, no programming is required to accomplish this. Thirdly, the setting of bias on programs leads to a reduction in the search space and thus may help to increase the chances of finding desired solution(s). We note that the reduction in search space may not only limit the space to searching valid programs as in STGP, it can also limit it so that the new language of programs is a sub-language of the original [Whi1995b, Whi1996]. Finally, the use of grammars facilitates the design of homologous operators, which will be discussed in the sequel.

The rest of this section is dedicated to an overview of the different grammar guided genetic programming system proposed in the literature.

2.4.1 Some Early Systems

Antonisse ([Ant1991]) appears to be the first researcher to propose the use of context-free grammars for generating grammar-based chromosomes in GAs. The proposed system, which is called grammar-based GAs, uses as chromosomes the strings from the language generated by a CFG G. When crossover is to be carried out between two chosen individuals, they have to be parsed into two derivation trees of G. Then, the crossover acts in a similar way as in the GGGP systems with tree based representation described in the next subsection. For each string, there might be more than one derivation tree for each string if the grammar is ambiguous. Therefore, in ([Ant1991]), it was suggested that the use of ambiguous grammars should be forbidden. Antonisse argued that this restriction was not problematic, as ambiguous grammars would cause slow convergence in any case. However, as pointed out in [NXH2002d], this may not be correct.

Some years after Antonisse proposed his grammar-based GA, a number of researchers suggested that grammars might be used to control the structure of programs in GP. Stefanski ([Ste1993]) proposed the use of abstract syntax trees to set a declarative bias for GP; Robston ([Ros1994]) demonstrated how a formal grammar might be used to specify constraints for GP in the context of engineering design; Mizoguchi and Hemmi ([Miz1994, Hem1994]) suggested the use of production rules to generate hardware language descriptions during the evolutionary process. Roston used the grammar for generating the initial population only, while the other works gave only a formal description. None of the systems maintained the derivation trees of the grammar, either re-parsing from the string (Stefanski) or combining with STGP (Roston).

2.4.2 Grammar Guided Genetic Programming with Tree-Based Representation

The first full-fledged grammar guided genetic programming (GGGP) systems were independently invented at about the same time by three different researchers. Whigham ([Whi1994, Whi1995a]) proposed his CFG-GP system, in which a context-free grammar was used to generate the population, which are derivation trees of the CFG. Schultz ([Shz1995]) derived his grammar guided genetic programming system for learning knowledge rules for expert systems. His system is similar to Whigham, differing mainly in the algorithm for initialising the population ([Boh1997]). Wong ([Won1994]), [Won1995a], [Won1995b], [Won1995c], [Won1995d], [Won1997]) proposed the LOGENPRO system, which uses definite clause grammars (DCGs), a type of logic grammars, in LISP to generate (logic and functional) programs to learn first order relations. DCGs are far more expressive than CFGs, being able to generate some context-sensitive languages. This is the primary difference between the systems; in other respects, LOGENPRO and CFG-GP are very similar. Given the similarity of the three systems, we base the description below primarily on one, CFG-GP [Whi1994, Whi1996]. The five basic components of a grammar guided genetic programming system are as follows:

Program Representation. Each program is a derivation tree generated by a grammar G (CFG for Whigham's and Schultz's systems, DCG logic grammar for Wong's system). Figure 2.10 depicts an individual in grammar guided genetic programming

Procedure for initialising population. In [Whi1995a], a simple algorithm

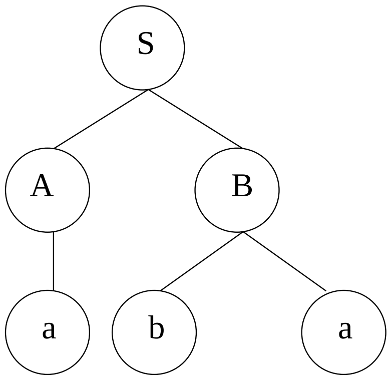


Figure 2.10: Derivation tree as programs in GGGP

was proposed to generate random derivation trees up to a depth bound, based on a procedure for labeling the production rules. Bohm and Schultz ([Boh1997] derived an algorithm for initialising the population based on the derivation-step-uniform distribution. The initialisation procedure LOGENPRO for was embodied in LISP inspired by the deduction mechanism of PROLOG [Won2003].

Fitness evaluation. Fitness evaluation is carried out on the individuals, i.e. derivation trees of G, in a similar way to standard GP.

Genetic Operators. The genetic operators are the selection mechanism, reproduction, crossover, and mutation. Selection and reproduction are as in standard GP. In crossover, two internal nodes labeled with the same non-terminal symbol of the grammar (G) are chosen at random, and the two sub-derivation tree underneath them are exchanged. Figure 2.11 depicts crossover in the GGGP system. We term crossovers exchanging sub-derivation trees stemming from the same non-terminal symbols homologous crossovers. Genetic recombination in biological systems is usually homologous, i.e. the genetic materials are not exchanged in completely random fashion but between genes that have similar function [Rid1996]. In the grammar guided genetic programming system, the metaphor of that homology is two sub-derivation trees starting with the same non-terminal symbol.

Mutation is performed by selecting an internal node at random. The non-terminal on this node is noted, and the sub-derivation tree rooted there is deleted. A new sub-derivation tree starting from the same non-terminal is randomly generated to replace the deleted one. Figure 2.12 shows how the mutation operator works.

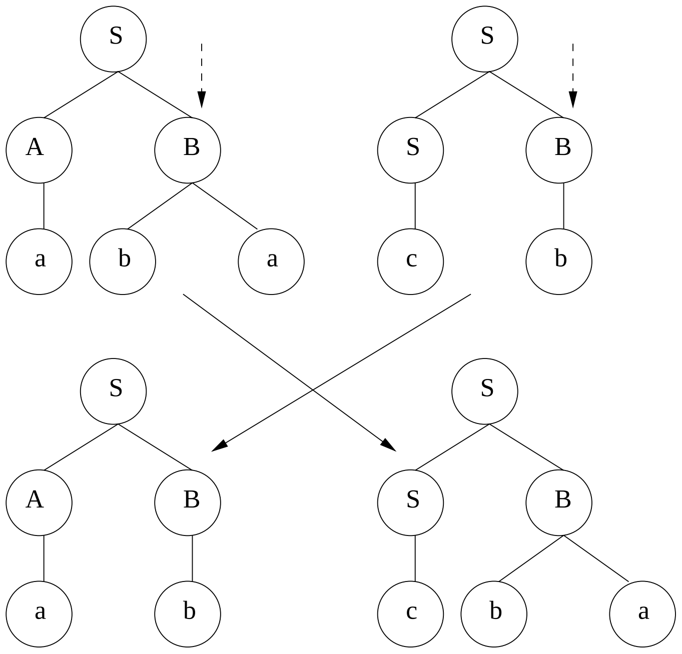


Figure 2.11: Crossover operator in GGGP

Parameters. Parameters include the population size (POPSIZE), maximal number of generations (MAXGEN), maximal depth for the individual (G derivation trees), and probabilities for the application of the operators.

Apart from the work by Whigham, Wong, and Schultz, there have been a number of similar grammar guided genetic programming systems, using derivation trees from a grammar as the representation for individuals. Gruau ([Gru1996]) presented some strong arguments for the use of context-free grammars as a tool

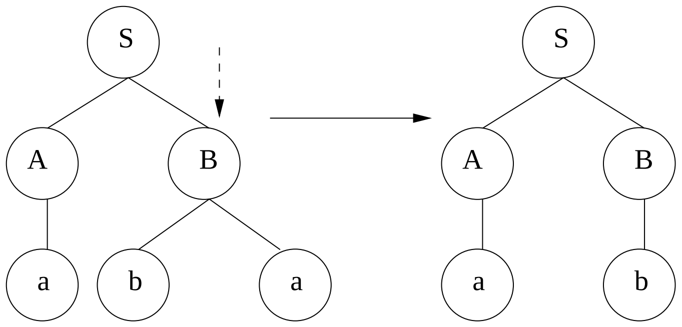


Figure 2.12: Mutation operator in GGGP

to set language bias on individuals. His resultant grammar guided genetic programming system is very similar to Whigham's CFG-GP. However, no depth was used to restrict the size of programs, potentially leading to code bloat problems. [Kei1999, Rat2000], introduced a grammar guided genetic programming system similar to Whigham's CFG-GP, but with a different initialisation process [Rat2000], to solve some industrial problems. The resultant system was known as dimensionally-aware genetic programming. Some recent versions of CFG-GP were also implemented in specific programming or web languages, in which the CFG is represented in Backus-Naur Form (BNF) [Tan2003a, Tan2003b, Tan2003c], [Mac2003].

During the past ten years, there have been a number of on-going projects on extensions to tree-based grammar-guided genetic programming, mainly by using formalisms that are extensions of, and therefore more expressive than, CFGs and/or logic grammars. These include using CFGs with linear constraints [Bru2002]; Stochastic Context-Free Grammars (SCFGs) [Whi1996], [Rat2001a], [Rat2002]; attribute grammars [Hus1998, Hus1999, Zva2004]; Definite Clause Translation Grammars (DCTG - an extension of the DCG grammars used in Wong's LOGENPRO system to incorporate semantics) [Ros2001].

2.4.3 Grammar-Guided Genetic Programming with Linear Representation

As in standard genetic programming, there have been a number of attempts to linearise the representation of programs in grammar guided genetic programming. The main methodology was to use a genotype to phenotype mapping, where the genotype is a linear sequence, and the phenotype is the derivation tree of the grammar (G). In the early work ([Kel1996],[Pat1996, Pat1997] [1],[Fre1998]), the genotype was a fixed string that is used to encode the index for derivation rules in G. The translation from a genotype to a phenotype is carried out from left to right, and the phenotype (G derivation tree) is built correspondingly. At each step, if there is still an uncompleted branch in the phenotype marked by a non-terminal A, a gene (number of bits) is read and interpreted as an integer number, indicating which, among the rules in the rule set P of G having left hand side A, will be used to extend that branch of the phenotype. If the phenotype is completed while there are still unused genes in the genotype, they are ignored (considered as introns). In the case that the translation uses all the genes, but the phenotype is still incomplete, some random or default sub-derivation trees will be used to complete the phenotype. The genetic operators are just those used in GAs.

The work on GGGP with linear representation that is closest to that in this thesis is grammatical evolution (GE) [Rya1998a, O'Ne2001b, O'Ne2003]. GE is an extension of the GGGP systems with linear representation as described above. Three innovations were added in GE: variable length, redundancy using the MOD rule, and the wrapping operation. The chromosome in GE has variable length rather than being fixed. Each gene is an 8-bit binary number, which is used to determine the rule for a non-terminal symbol when it is expanded. If the decimal number represented by the gene is bigger than the number of rules for the non-terminal, the modulo operation is used to calculate the rule number.

[1]Paterson has more recently extended his system to handle context-sensitive grammars [Pat2002]

For example, suppose the non-terminal symbol A is currently being expanded, and A has 3 rules; if the next gene in the translation process is 7, then the rule used to expand A is 7 MOD 3= 1. The wrapping operation is used when the translation from genotype to phenotype has run out of genes when the phenotype is still incomplete. The translation process starts to re-use the gene from left to right. If the number of wrappings exceeds a predetermined maximal bound, then the translation finishes and the (invalid) individual is assigned a very low fitness. The following is an example of the GE genotype to phenotype mapping process. Suppose we have a CFG $G = (\sum, N, P, EXP)$, where $\sum = \{+, -, x, y\}$ is the set of terminal symbols, $N = \{EXP, OP\}$ is the set of nonterminal symbols, and the rule set P is as follows (definition of context-free grammars is given in appendix A)

$0: \; EXP \rightarrow EXP \; OP \; EXP$

$1: \; EXP \rightarrow X$

$2: \; EXP \rightarrow Y$

$0: \; OP \rightarrow +$

$1: \; OP \rightarrow -$

Figure 2.13 shows an example of genotype to phenotype mapping in GE with the above grammar. For convenience, decimal rather than binary numbers are depicted.

Since it was proposed, there have been a range of on-going researches to develop,

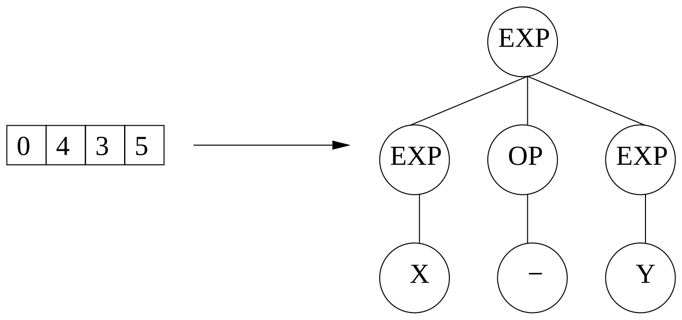


Figure 2.13: An example of GE genotype-phenotype mapping

extend, and apply GE in many ways, including a study of the effect of GE crossover on the phenotype [O'Ne2000a, O'Ne2001a, Kei2001], alternatives to the MOD rule in genotype-phenotype translation [Kei2002], different search strategies [O'Su2002], new representations based on the GE representation aiming to reduce the effects of positional dependence [Rya2002a], implementation of GAs through GE using an attribute grammar [Rya2002b], and so on.

There are two key problems with the GE representation. Firstly, a valid genotype may code for an infeasible (not fitness calculable) phenotype. In the above example, a genotype such as "0310" cannot be decoded, because the translation will never finish (it produces unbounded derivation trees of the infinite expression (X-(X-(X-(X-...)))). Although the problem can be treated by assigning these individuals very low fitness values, it can become an obstacle for evolution and search on the GE representation if the proportion of genotypes coding for infeasible phenotypes is large [2].

The second problem with GE is that, as with GEP, it does not fulfill the causality principle. Although there are operators that make small (bounded) and controllable changes on the genotype, the resulting changes on the phenotype are unpredictable (and uncontrollable). A small change in a gene at one position may completely change the expressiveness (coding or non-coding), or even the meaning (if there is more than one non-terminal in the grammar) of all the genes following that position. In the extreme, it may change the corresponding phenotype from feasible to infeasible. For instance "011100001" in the preceeding grammar codes a valid phenotype; if gene 4 mutates to "0" the genotype becomes "01100001", whose phenotype is infeasible.

[2]Since the mapping is redundant, a genotype like "0310" may have many equivalents - in this case, 3310 etc.

2.4.4 Some Problems with Grammar-Guided Genetic Programming

As with GP, grammar-guided genetic programming is still far from resembling GAs despite some significant efforts in this direction. The problems are similar as those discussed in section 2.3.2.

In tree-based GGGP systems [Whi1994, Won1994, Shz1995], it is even more difficult than in GP to design operators that can make small and bounded changes. In addition to the constraints imposed by the GP tree grammar, the derivation trees in GGGP are even more constrained by the rewriting rules of the grammars. Therefore, a change to a program, which is the derivation tree of the grammar [Whi1994, Won1994, Shz1995], could easily result in an invalid program (invalid derivation tree). The change caused by the operators implemented in those systems so far (sub-derivation tree crossover and sub-derivation tree mutation), are not controllable as they could destroy large sub-derivation trees underneath the chosen points for genetic operation. It is also difficult to implement bio-inspired operators in those GGGP systems. We argue that the reason is not only the tree-based nature, as in GP, of the derivation trees used in tree-based GGGP systems for representing programs, but also it is the set of rewriting rule constraints it must conform.

For GGGP systems with linear representation [Kel1996, Pat1996, Rya1998a, O'Ne2001b, O'Ne2003] (including, we believe, the recent extensions of GE), the problems are only partly solved in the sense that it is possible to design operators that could make small and bounded changes and operators that are bio-inspired in the genotype space. However, the non-causality in their genotype-phenotype mapping means that the effect of small and bounded changes disappears in the phenotype space. Furthermore, as argued later in the thesis, even the bio-inspired operators could be designed in the genotype space, the context for using those operators might be vague, since the effect on the phenotype space of those operators might be unpredictable.

The lacking of operators that could make small and bounded changes could create a number of problems for GGGP. For example, the absence of such operators means GGGP systems have not had any powerful tool to explore the neighbourhood structure of their syntactically-constrained search spaces. Therefore, it is difficult to conduct fitness landscape study on the search spaces of GGGP systems. Fitness landscape analysis not only helps to reveal the search difficulty of the problems but also helps to investigate the effect of changing language bias (grammars) on the characteristics of the syntactic-constrained search spaces as shown in chapter 9 of the thesis. The lacking of bio-inspired operators makes GGGP are very operator-poor (almost all of tree-based GGGP systems have only two genetic operators mentioned above). The usefulness of bio-inspired operators are demonstrated in chapter 6 of the thesis.

2.4.5 Applications of Grammar-Guided Genetic Programming

For the last decade, grammar-guided genetic programming systems have been used to successfully solve a substantial number of academic and real-world problems, including learning control knowledge for planning [Ale2001]; evolving neural networks [Tsa2002]; composing music [Put1996, Pue2002]; Predicting time series [Whi2001a], [Whi2001b], [Bra2002]; estimating software size [Sha2002]; accomplishing data mining task [Fre2002], [Won2000]; analysing biological sequences [Ros2002]; and solving problems in industry [Rat2001b].

2.5 Conclusion

In this chapter, we surveyed a range of works on representation for genetic programming and grammar-guided genetic programming. We raised the issues of preserving topological structures and defining operators that make small and bounded changes. We pointed out that there is a trend toward linearising tree-based structures in GP and GGGP, in order to increase its similarity to GAs and solve these problems. However, most of the linear representations for GP and GGGP proposed in the literature do not possess the causality property. Therefore, the problems persist in the phenotype space.

In the succeeding chapters, a new representation for GP and GGGP is proposed and analysed. The new representation is tree-based but flexible, facilitating the design of bio-inspired operators making small and bounded changes. Moreover, its genotype to phenotype mapping satisfies the causality property, so that the changes are also small and bounded on the phenotype space. Those properties turn out to be very useful, as shown in subsequent chapters of the thesis.

Chapter 3

A Tree Adjoining Grammar Based Representation for Genetic Programming

This chapter is intended as a foundation for the thesis, laying out a new representation for genetic programming (grammar-guided genetic programming), namely tree adjoining grammar based representation (TAG-based representation). It starts with some concepts of tree adjoining grammars and tree languages. In particular, a brief description of some current versions TAG derivation tree is given, followed by our variant of the concept of TAG derivation trees, specifically optimised as a GP representation. Next, we give an overview of some general representation principles and guidelines from the literature of evolutionary algorithms. Based on these, we define a TAG based representation and discuss the potential benefits of TAG based representation for genetic programming (grammar-guided genetic programming). Finally, the conclusion of the chapter lays out how subsequent chapters will provide more detailed justification for the use of TAG-based representation.

3.1 Tree Adjoining Grammars

Tree adjoining grammars (TAGs) are tree-generating and analysis systems, first proposed by Joshi et al in [Jos1975]. Tree Adjoining Grammars (TAGs) have become increasingly important in Natural Language Processing (NLP) since their introduction.

The aim of TAGs is to more directly represent the structure of natural languages than is possible in Chomsky languages, and in particular, to represent the process by which natural language sentences can be built up from a relatively small set of basic linguistic units by inclusion of insertable sub-structures. Thus 'The cat sat on the mat' becomes 'The big black cat sat lazily on the comfortable mat which it had commandeered' by the subsequent insertion of the elements 'big', 'black', 'lazily', 'comfortable', 'which it had commandeered'. In context-free grammars (CFG)(Chomsky's formalisms of type 2), the relationship between these two sentences can only be discerned by detailed analysis of their derivation trees; in TAG representation, the derivation tree of the latter simply extends the frontier of the former. To put it another way, the edit distance between the derivation trees of these closely related sentences is much smaller in TAG representation than in CFG representation.

Furthermore, one of the primary motivation for TAGs is to reply to the challenge in natural language processing for building a formalism to capture the long distance relationships between sub-structures such as subject-verb agreement. This capture is required even when the distance might change arbitrarily such as in 'wh-movement' phenomenon (given in 3.1.3). By using elementary trees (defined in the next subsection) rather than flat rules as in CFGs, TAGs can extend the domain of locality to cover through several equivalent rules level in CFG derivation trees; and, by using the adjunction operation (defined in the next subsection), TAGs can simulate the moving relationships between the subcomponents of sentences as in 'wh-movement' phenomenon.

In the initial formulation, the only tree rewriting operation in TAGs was adjunction (described below) and the formalisms were called tree adjunct grammars. After a sequence of developments in [Jos1977, Jos1985, Jos1987, Jos1991], a new operation called substitution (described below) was added, and the formalisms were re-named tree adjoining grammars.

Since TAGs (tree adjunct grammars and tree adjoining grammars), from their only days were shown to possess a number of invaluable properties for handling various issues in natural language processing [Jos1985, Kro1985, Kro1987]. Therefore, from the mid-1980s, there has been a surge in the study and applications (mainly in natural language processing) of TAGs. An entire international journal issue was devoted to the study of TAGs [SPECI1994] and a project was dedicated to develop a tree adjoining grammar for English [XTA1995]. A recent comprehensive overview of TAGs can be found in [Jos1997].

In the next four subsections, some basic concepts of TAGs and TAG derivation trees is given.

3.1.1 Definitions of Tree Adjoining Grammars

In this subsection, a number of the basic definitions of tree adjoining grammars are given. Firstly, the standard definition of tree adjoining grammars (and their lexicalised version) from [Jos1997], and then the concepts of their languages (both tree and string), is defined.

Definition 3.1 (Tree Adjoining Grammars)

A tree adjoining grammar is a tree-rewriting system comprised of a quintuple $(\sum, N, I, A, S)$, where:

(i) $\sum$ is a finite set of terminal symbols.

(ii) N is a finite set of non-terminal symbols: $N \cap \sum = \emptyset$.

(iii) S is a distinguished non-terminal symbol: $S \in N$.

(iv) I is a finite set of finite trees, called initial trees (or α-trees). In an initial tree, all interior nodes are labeled by non-terminal symbols, while the nodes on the frontier are labeled either by terminal or non-terminal symbols. Non-terminal symbols on the frontier of an initial tree are marked as $\downarrow$ for substitution.

(v) A is a finite set of finite trees, called auxiliary trees (or β-trees). In an auxiliary tree, all internal nodes are labeled by non-terminal symbols. A node on the frontier is labeled either by a terminal or a non-terminal symbol, and all nodes on the frontier labeled by non-terminal symbols, except for one special distinguished node, are marked as $\downarrow$ for substitution. That distinguished node is known as the foot node. The foot node must be labeled by the same non-terminal symbol as that of the trees root node, and is usually marked with an asterisk (*). The trees in $E = I \cup A$ are called elementary trees. Initial trees and auxiliary trees are denoted α and β respectively. A node labeled by a non-terminal (terminal) symbol is sometime called a non-terminal (terminal) node. A tree with a root labeled by a non-terminal symbol X is called an X-type elementary tree. In essence, an α-tree with all terminal symbols on its frontier is just like a minimal complete sentence, while a β-tree is a minimal recursive structure used to modify complete sentences (by using adjunction described below).

An example of TAGs. $G_1 = \{\sum, N, I, A, S\}$, where $\sum$ is a set of English words, $N = \{S, VP, NP, V\}$ and $E = I \cup A$ as in Figure 3.1.

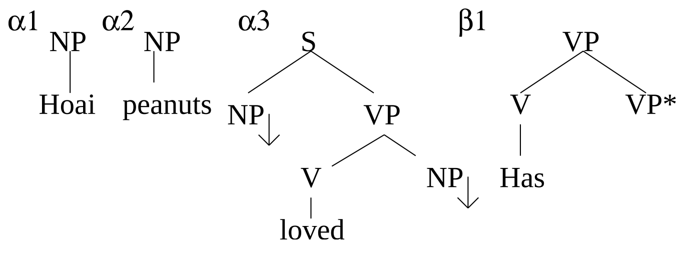


Figure 3.1: A simple TAG for some English sentences

The key operations used with tree-adjoining grammars are the adjunction and substitution of trees. Adjunction builds a new (derived) tree γ from an auxiliary tree β and a tree λ (initial, auxiliary or derived). If a tree has an interior node labeled A, and β is an A-type tree, the adjunction of β into λ to produce γ is as follows. Firstly, the sub-tree λ_1 rooted at A is temporarily disconnected from λ. Next, β is attached to replace the sub-tree. Finally, λ_1 is attached back to the foot node of β. γ is the final derived tree achieved from this process. Adjunction is illustrated in Figure 3.2.

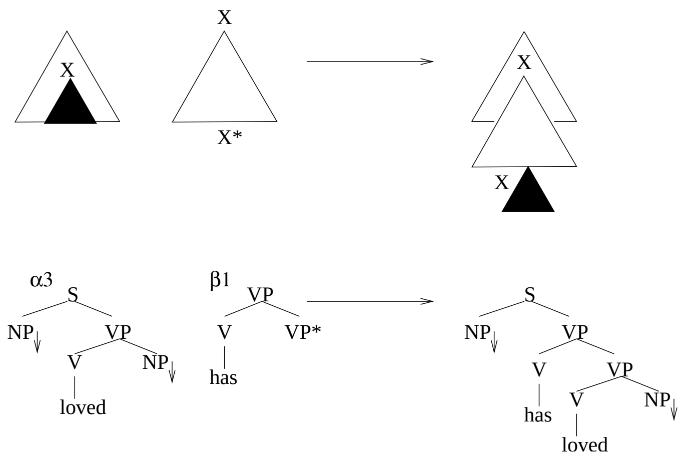


Figure 3.2: Adjunction Operation.

In substitution, a non-terminal node on the frontier of an elementary tree is substituted by another initial tree with a root labeled with the same non-terminal symbol. Substitution is illustrated in Figure 3.3.

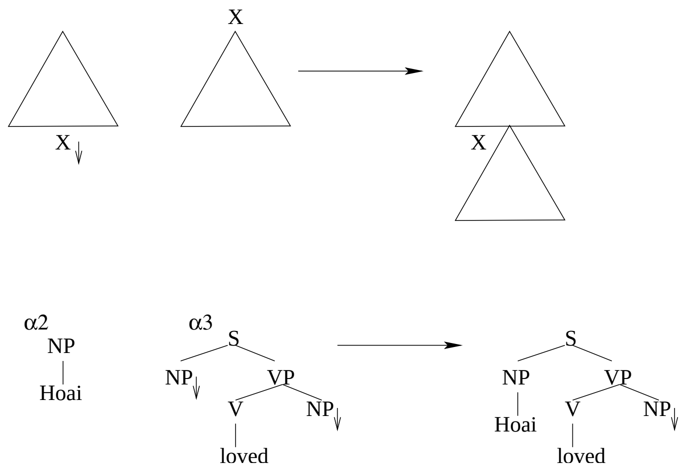


Figure 3.3: Substitution

The tree set (tree language) of a TAG G can be defined as follows:

Definition 3.2 (Tree Set of TAGs)

T_G = { all trees t / t is completed and t is derived from some initial S-trees through some adjunctions and substitutions }.
where a tree t is completed if all of the leaf nodes of t are labeled by terminal symbols. The string language generated by G is defined as follows:

Definition 3.3 (String Set of TAGs)

LG = { $w \in \sum^*$ / w is the yield of some tree $t \in T_G$ }.
where the yield of a tree is the string composed by reading off all the leaves of the tree in left-to-right order.

We previously noted that in the early days [Jos1975, Jos1977, Jos1985], the only operation in TAGs was adjunction. Later, substitution was added. The addition of substitution does not change the generative power of TAGs. In other words, it is possible to use TAGs with adjunction as the sole operation to generate all the tree and string sets which can be generated by TAGs using both adjunction and substitution [Jos1997]. However, the addition of substitution, in practice, helps to make the formalism more compact, by reducing the size of its elementary tree set.
A special type of TAG known as a lexicalised TAG (LTAG) can be defined as follows [Jos1997]

Definition 3.4 (Lexicalised TAGs)

A lexicalised tree adjoining grammar (LTAG) is a tree adjoining grammar (TAG) satisfying the requirement that each of its elementary trees contains a terminal node.

Although there are more constraints on LTAGs, it can be shown that LTAGs are equivalent to TAGs, in the sense that LTAGs are capable of generating any TAG tree or string set. In this thesis, we will subsequently only need to use LTAGs. So we will abbreviate by using the terms tree adjoining grammars (TAGs) and lexicalised tree adjoining grammar (LTAGs) interchangeably. We will usually denote an LTAG with the notation G_{lex}.

3.1.2 Derivation Trees in Tree Adjoining Grammars

In TAGs, there is a distinction between the derivation and the derived tree, where the former encodes the history of adjunctions (and substitutions) used to generate the latter. There are at least three definitions of TAG derivation trees in the literature, given in [Sha1987, Wei1988], [Sch1994], and [Jos1997]. After they are given and discussed briefly below, we will present our own variant specialised for our application as a GP representation.

Standard Definitions of TAG-derivation trees

The concept of TAG derivation trees was first given in [Sha1987], and clarified in [Wei1988]. The following definition of TAG-derivation trees is adapted from [Wei1988]. We note that, at the time Shanker, and subsequently Weir, defined the concept of TAG derivation trees, the only operation they used in TAGs was adjunction.

Definition 3.5 (Weir's conception of TAG derivation trees) A TAG derivation tree is a labeled object tree, characterised as follows. The root node is labeled with the name of an S-type initial (α) tree in the elementary tree set; the nodes other than the root are each labeled with the name of an auxiliary (β) tree in the elementary tree set. Each link between a parent and a child node is labeled with an index number, indicating the place in the elementary tree represented by the parent node to which the auxiliary tree represented by the child node is to be adjoined. For instance, suppose the parent node is labeled γ_1 (an elementary tree of the grammar), the child node is labeled β_1 (an auxiliary tree of the grammar), and the link between the two nodes is labeled with an integer 5. Assuming that all the nodes of an elementary tree are indexed by an integer, namely its occurrence position when the elementary tree is traversed in preorder, the two above nodes and the link between them mean, when constructing the derived tree from the derivation tree, that an adjunction of β_1 into γ_1 is carried out at node 5 (in preorder traversal) of γ_1.

Figure 3.4 depicts Weir's concept of a derivation tree in TAGs. We note that the translation (decoding) of a derivation tree into a derived tree proceeds in a bottom-up manner using adjunctions.

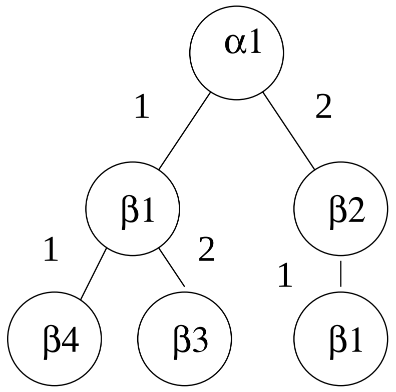


Figure 3.4: Weir's TAG derivation tree

In Weir's derivation tree, each adjoining address (node) permits only one adjunction. In other words, all the labels on the links from a parent node to each of its child nodes are unique. Thus the order of adjunctions at each node is not important. However, in practice, it is more convenient if this order is fixed (for example from left to right). If a fixed order of adjunction is employed, the derivation tree is said to be in canonical form. From this point on, we will assume that all derivation trees are in canonical form (i.e. that there is a fixed order for carrying out adjunctions). Consequently, for each TAG derivation tree, there is only one way to translate (decode) it into the corresponding derived tree.

TAG-derivation trees defined as above have a rather interesting property which, to the best of our knowledge, has not been discussed and/or used in the field of natural language processing. Since (Weir's) TAG derivation tree is an object tree, it is possible to remove any subtree, and the resultant tree is still a perfectly valid TAG derivation tree (so its derived tree is still a complete derived tree). In other words, the arity (number of children) of each node in a TAG derivation tree is not fixed. For instance, suppose that the elementary tree γ labels a node a in a TAG derivation tree t. Moreover, assume that γ has k

available adjoining addresses. Then, a can have j children, for any $0 \leq j \leq k$. This property of (Weir) TAG derivation trees we call the non-fixed arity property. The usefulness of this property will be subsequently discussed and shown in this thesis.

Definition 3.6 (Schabes-Shielber conception of TAG derivation trees) The definition of TAG derivation tree in [Sch1994] is essentially the same as Weir's, with the exception that it permits multiple adjunctions at each adjoining address. Figure 3.5 depicts a Schabes-Shielber TAG derivation tree.

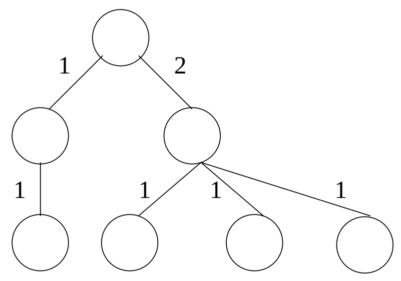


Figure 3.5: Schabes-Shielber TAG derivation tree

Despite the extension in allowing multiple adjunctions at one address, it was shown in [Sch1994] that the Schabes-Shielber definition of derivation tree is exactly equivalent to Weir's. Although it is argued in [Sch1994] that the admission of multiple adjunctions at each address makes TAG derivation trees more relevant to linguistics, it has an important limitation for our purposes. Allowing multiple adjunctions at an address can potentially lead to an unbounded-arity phenomenon in the TAG derivation tree (i.e. there is no limit to the number of children which might be adjoined at a given node). There is a strong possibility that unbounded arity would increase the complexity of GP search.

Definition 3.7 (Joshi-Schabes conception of TAG derivation trees) The root of a derivation tree is labeled by an S-type initial (α) tree. All other nodes are labeled by auxiliary (β) trees in the case of adjunction, or initial (α) trees in the case of substitution. A tree address is associated with each node (except the root node) in the derivation tree. This tree address is the node in the elementary tree labeling the parent node on which the adjunction or substitution is performed. Figure 3.6 depicts such a TAG derivation tree, in which the unbroken lines denote adjunction and the broken lines denote substitution.

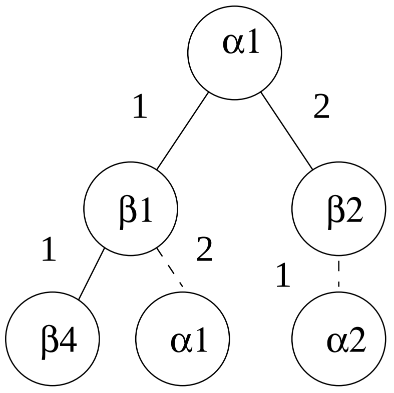


Figure 3.6: Joshi-Schabes TAG derivation tree

In essence, Joshi-Schabes conception of TAG derivation trees [Jos1997] is the natural extension of Weir's to include substitution. However, we note that Joshi-Schabes' TAG derivation trees have lost the non-fixed arity property satisfied by Weir's. Although the derivation tree is still valid when any subtree is removed, the derived tree might not remain completed if an initial tree used for substitution is deleted (e.g if in figure 3.6, $\alpha 1$ is deleted). Nevertheless, Joshi-Schabes definition of TAG-derivation tree is equivalent to Weir's definition, since it is always possible to use TAG with adjunction only [Jos1997].

A Special Form of TAG-derivation Trees

As discussed above, the Schabes-Shielber conception of TAG derivation tree raises the problem (from a GP perspective) of unbounded arity. On the other hand, Weir's conception possesses a potentially useful property, namely non-fixed arity. However Weir's TAG conception does not allow substitution. To use Weir's form, one must use a TAG without substitution, in practice resulting in a very verbose formalism. Joshi and Schabes incorporated substitution in their definitions but in the process lost the non-fixed arity property.

In order to maintain the non-fixed arity property, while incorporating substitution in TAG derivation trees, we give a new formulation of TAG derivation trees below.

Definition 3.8 (TAG derivation tree with restricted substitution)
A TAG derivation tree is defined as in definition 3.7, except that no adjunction is permitted at a node representing an initial tree used for substitution (in other words, this ensures that all adjunctions occur before any substitutions are attempted).

In essence, our conception of TAG derivation tree is very similar to Joshi-Schabes. The only difference is that if a node is labeled by an initial tree, and the link to its parent node indicates that the operation is substitution, then this node must not have any children. With this restriction, we can regard substitution as an in-node operation. In other words, our form of TAG derivation tree is a tree of objects in which the links between them indicate adjunction (at specified) addresses, and each node has an attached list of initial trees (called lexemes) to be substituted into opened nodes (called lexicons) in the elementary tree labeling this node. Since the type of the derivation tree is a object-tree, the non-fixed arity property in Weir's definition is maintained, while it also facilitates the substitution operation for compacting the formalism just like in Joshi-Schabes derivation trees. Figure 3.7 shows the structure of this type of TAG derivation tree. The places for the substitutions of $\alpha 1$ and $\alpha 2$ into $\alpha 3$ are determined by the preorder tree traversal of $\alpha 3$ (i.e at N and at V).

It should be noted that our new formulation of TAG derivation tree does not change the generative power of TAGs, since TAG can work without substitution (and hence with restricted substitution). In practice, our type of TAG derivation tree can be realised simply by designing TAGs with no adjoining addresses in initial trees that are used for substitution.

Notational conventions:

- From now on, we use the term "TAG derivation tree" to denote the form defined in definition 3.8.

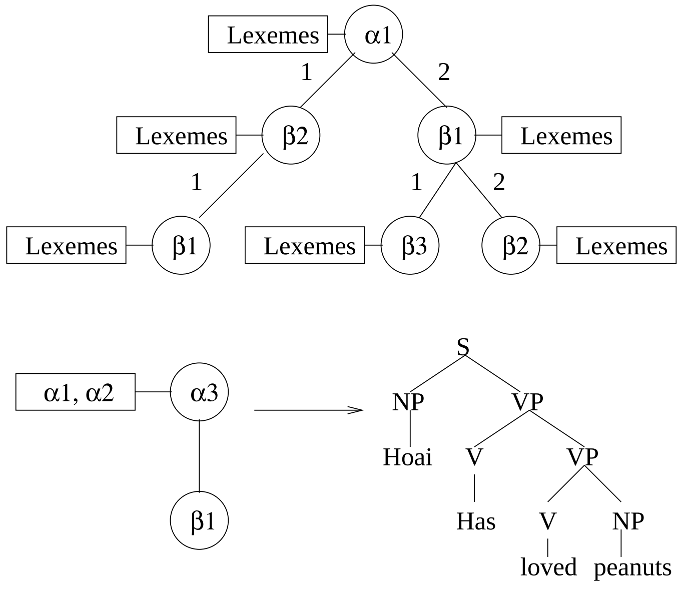


Figure 3.7: Example of the new form of TAG derivation tree.

- Since the substitution is an in-node operation in this representation, it is often omitted (e.g. in figures) where this does not cause confusion.
- The initial trees used for substitution (i.e. the list attached to nodes of a TAG derivation tree as in Figure 3.7) are called lexemes. The locations into which lexemes are substituted are known as lexicons.
- For the sake of brevity and convenience, we use the phrase "node n" in a TAG derivation tree to mean the elementary tree node labeled by n.

3.1.3 Some Properties of TAGs

Apart from the non-fixed arity property of their derivation trees, TAGs have a number of other interesting properties, especially from a linguistic perspective. A comprehensive survey can be found in [Jos1997]. In this subsection, we restrict our attention to some properties of TAGs that are relevant to GP. We simply state their relevance here, and explore them in more detail later in the thesis.

Regarding the generative power of TAGs, we have the following:

- The TAGs string languages strictly include CFG languages, and are strictly included in indexed languages.

- The set of CFG derivation trees is strictly included in the set of TAG tree languages.

- For every context-free grammar G, there is an LTAG G_{lex} such that the tree language of G_{lex} is the set of G derivation trees. G_{lex} is said to strongly lexicalised G.

- A number of algorithms for finding G_{lex} for a given CFG G are proposed in [Sch1990], [Sch1993a], [Sch1993b], [Sch1995], and [Jos1997]. The algorithm given in appendix A and used in this thesis is from [Sch1990]. We note that this algorithm is based on the ideas of separation between the recursive parts (structure) and non-recursive parts (lexicon) of the CFG G.

- The only operation necessary in the resultant LTAG (G_{lex}) is adjunction. However substitution can be added to make the elementary set more compact. Moreover, it is possible to restrict the substitution trees to the non-recursive parts of the grammar. In so doing, the initial tree used for substitution can not be adjoined by other auxiliary trees.

- G_{lex} derivation trees are represented as our special form of TAG derivation tree defined in the previous subsection.

Two very important characteristics of TAGs have been emphasized in the linguistic literature, namely the extended domain of locality (ELD) and factoring recursion from the domain of dependencies (FRD) [Jos1997].

Extended domain of locality (ELD). In CFG, the domain of dependencies between lexicons is restricted to one (string) rewriting rule. Consequently, it is difficult for CFGs to represent the long-distance dependencies between lexicons that extend over several grammar rules, for example English subject-verb agreement. In TAGs, since the primitives of the formalism are elementary trees, which can incorporate several CFG rules, the domain of dependencies is generally larger, so that useful dependencies between lexicons in an elementary tree can be specified. ELD means that TAGs can capture long-distance dependencies that are difficult to represent with CFGs.

Factoring recursion from the domain of dependencies (FRD). As well as capturing specific long-distance dependencies directly into elementary trees, TAGs can also capture expandable long-distance dependencies. The expandable long distance dependencies are an important phenomenon in natural languages such as English. An example is the well-known "wh-movement" [Gri1986, Mol1988, Bar1998]. For instance, the long-distance dependency between "who" and "like" in "who did Maria like?" can be factored into an even longer subjacency as in "who did Bill say that Maria likes?". In TAGs, the adjunction operation helps to factorise the dependencies captured in the elementary trees, since it can increase the subjacency distance between the lexicons [Jos1997].

The following example illustrates ELD and FRD in an LTAG.

An example of ELD and FRD in TAGs. We give a simple context-free grammar G for a fragment of English as follows. $G = (N, \sum, P, S)$, where N={S, NP, VP, ADVP, ADV, V, NP}, $\sum$= {Hoai, I, You, They, We, likes, like, peanuts, passionately, absolutely, really,...}. The (string) rewriting rule set P is:

Syntactical rules

1. $S \rightarrow NP\ VP$
2. $VP \rightarrow ADVP\ V\ NP$
3. $VP \rightarrow V\ NP$
4. $ADVP \rightarrow ADVP\ AV$

Lexical rules

5. $NP \rightarrow Hoai|I|You|We|They$
6. $NP \rightarrow peanuts$
7. $V \rightarrow likes|like$
8. $ADV \rightarrow really|absolutely|passionately$

We apply the algorithm in [Sch1990, Jos1997] (given in Appendix A), to generate G_{lex} strongly lexicalising G: $G_{lex} = (\sum, N, I, A, S)$, where $\sum$ and N are as in G.

The elementary tree set $E = I \cup A$ is given in Figure 3.8 (the lexicons L1 and L2 are to be substituted with lexemes using the above lexical rules; i.e L1 can be replaced with one lexeme in the left part of rule 5, and L2 can be replaced with one lexeme in the left part of rule 8).

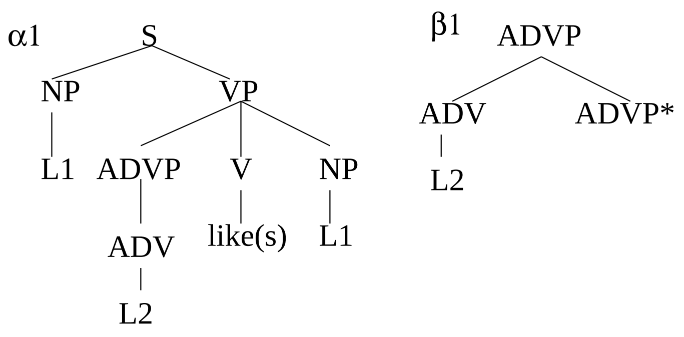


Figure 3.8: Another simple TAG for some English sentences

The sentence “Hoai passionately likes peanuts” exemplifies ELD: Figure 3.10 shows its derivation tree and derived tree. The dependency is the subject-verb agreement between “Hoai” and “likes”. We note that it is not possible to enforce this dependency in G, since it stretches through a number of the rule levels. Since the domain of dependency of G_{lex} consist of several rule levels, it is much larger than that of G, which is limited to one rule level. So it is easier for G_{lex} to capture the long distance dependency than is is for G.

The effect of FRD can be seen by adjoining $\beta 1$ at node ADVP, giving “Hoai absolutely passionately likes peanuts”. Again, this effect cannot be reproduced in the CFG.

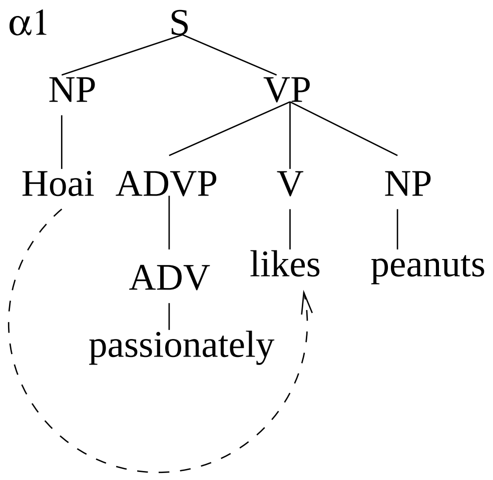


Figure 3.9: An example of ELD in TAGs

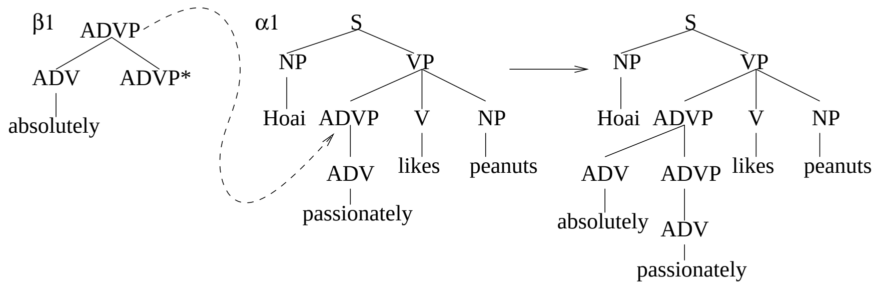


Figure 3.10: An example of FRD in TAGs

3.2 Tree Adjoining Grammar Based Representation for Genetic Programming

3.2.1 Representation in Evolutionary Algorithms

In the field of evolutionary algorithms (EAs), the representation is the first decision a designer has to make. For example, the differences between the genetic programming systems discussed in the previous chapter lay primarily in their representations. Although representation is such a vital issue for EAs, there is only limited theory on what makes a good representation [Rot2002], though a number of general guidelines or principles have been proposed.

Goldberg's Principles of representation. Goldberg [Gol1989], proposed two very early general principles for designing a representation, the principles of meaningful building blocks and of minimal alphabets:

- Principle of meaningful blocks: The schemata should be short, of low order, and relatively unrelated to schemata over other fixed positions.
- Principle of minimal alphabets: The alphabet of the encoding should be as small as possible, while still allowing a natural representation of solutions.

From analysing representation issues, Palmer and Kershenbaum in [Pal1994a, Pal1994b, Pal1994c] formulated general guidelines for representation in evolutionary algorithms [Pal1994b]:

- A representation should be able to represent all possible phenotypes.
- A representation should be unbiased in the sense that all possible individuals are equally represented in the set of all possible genotypic individuals.
- A representation should not permit infeasible solutions.
- The process to translate a genotype to its corresponding phenotype should be simple.
- A representation should possess locality (or causality). Small changes in the genotype should result in small changes in the phenotype.

A few year later, Ronald ([Ron1997]) surveyed a number of principles for representation in genetic algorithms from the literature and summarised the following set of guidelines:

- Representation should be adjusted to a set of genetic operators so that the building blocks are preserved from the parents to the offspring.
- Representation should minimise epistasis.

- Feasible solutions should be preferred.
- The problem should be represented at the correct level of abstraction.
- Representation should exploit an appropriate genotype-phenotype mapping process, if a simple mapping to the phenotype is not possible.
- The phenotype should not be represented by more than one genotype.

In [Rot2002], a framework for analysing an EA representation was proposed. A representation can be investigated from three perspectives, namely redundancy, building block scaling, and distance distortion.

The redundancy perspective studies whether a representation's genotype-phenotype mapping is many-to-one, and whether any redundancy is useful. As demonstrated in [Shi1999, Sha2000, Shi2000], redundancy is useful if it can form extended and connected neutral walks on the genotype space. By drifting along this neutral walk, an evolutionary search can escape from local optima. In [Sha2000, Rot2002], it was shown that uniform redundancy is harmful for EAs. Thus Ronald's principle 6 should be expanded with "for redundant genotype-phenotype mappings, keep in mind that some will be useful whereas other might be harmful".

The building block scaling perspective studies how a representation changes the size of primitive building blocks for solving a problem.

The last perspective, distance distortion, considers the degree of distortion a genotype-phenotype mapping introduces into the mapping between their distance metrics. It was argued in [Rot2002] that a low distance distortion is desirable. Moreover, it is argued in [Rot2002] that achieving a low distance distortion, requires the locality property, i.e. Palmer's last principles of representation. We note that the locality property is perhaps the most thoroughly studied in the EA literature, and has been thoroughly validated throughout the subfields of EA as one of the most important properties that a representation should possess [Rec1973, Ros1995, Sen1997, Ige1998, Dro1998, Got1999, Sta2000, Leh2003].

For most EA fields, the representation imposes only very limited restrictions on the design of operators; however as discussed in the previous chapter, this is not generally the case for GP and GGGP. Hence we add a further principle, namely that it is desirable for a representation that it facilitates the design of a wide range of search operators, especially those which cause bounded and controllable variation. In the next subsection, we introduce a new representation for GP (GGGP). We aim to investigate throughout this thesis the extent to which it satisfies the principles above, and to show that it possesses a number of the proposed useful properties.

3.2.2 TAG-based representation

In this section, a new representation, namely TAG-based representation, for genetic programming (GGGP) is introduced. We discuss its compliance with the preceding principles, and give some arguments why TAG-based representation might be desirable, indicating how the detail is provided in the rest of the thesis.

Our TAG-based representation for GP (GGGP) uses the LTAG derivation trees from the previous section as the representation for the GP genotype. The problem domain is described by an LTAG grammar, G_{lex}, just as in [Whi1995b, Whi1995c], it is constrained or biased by a Chomsky grammar (CFG or otherwise). Since TAGs can generate the tree set of context-free languages and of some context-sensitive languages, the TAG-based representation can be used to solve problems in the domain of context-free as well as of some context-sensitive languages as in [Won1997].

While the search of the problem domain is conducted on TAG derivation trees, the fitness evaluation is carried out on the derived trees decoded from them. Therefore, the TAG-based representation has a natural genotype-phenotype mapping, whereby genotypes are TAG derivation trees and phenotypes are their corresponding derived trees (or course, just as in GGGP, there is often a further mapping from the derived tree to a term tree or other phenotype, so that the derived tree is only an intermediate phenotype). A grammar guided genetic programming (called TAG3P) using this representation is designed in the next chapter.

While Palmer's first four general principles are relatively straightforward, it may not be obvious that the TAG representation satisfies Palmer's locality property. In more mathematical terms, this requires us to prove that the mapping from TAG derivation trees to derived trees is Lipschitzian [Rud1976] (described below). However, before proving this, it is necessary to discuss potential metrics in the derivation tree and TAG derived tree spaces.

In the GP literature, a number of researchers ([Eka2002],[Cle2002], [Van2003a], [Van2003b], [Eka2004]) have used the tree-alignment metric to measure the difference between two labeled trees. To compute the tree-alignment metric, the two trees are aligned top-down from the root, and the differences (in terms of label, arity) between nodes are accumulated by certain calculating functions. However this metric, while it has some useful properties, corresponds poorly to our intuition of labeled tree distance. For instance, if the difference between two trees t_1 and t_2 is only at the root node, (e.g t_1 is formed from t_2 by adding a new root node), the tree alignment distance between them is large while our intuition views them as similar (i.e the distance between them should be small). Figure 3.11 depicts the situation in this example.

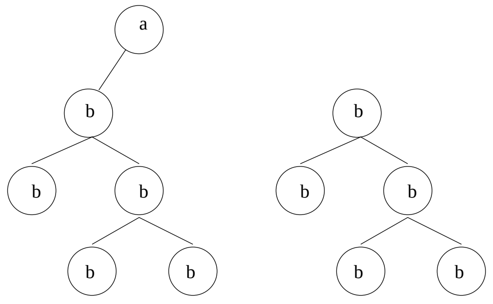


Figure 3.11: An example of the conflict between tree alignment metric and the intuitive sense of the similarity between two trees. The left and the right trees are very similar in the intuitive sense but far from each other using tree alignment metric

Consequently, the tree metric adopted here is the tree edit distance proposed in [Lu1979] and used elsewhere in genetic programming [ORe1997]. In [Lu1979], the tree edit distance between two labeled trees is defined as the length of the shortest sequence of editing operations that transforms one tree to another. The editing operations are deleting a node, inserting a node, or changing the label of a node. With edit distance as the metric, the mapping between derivation and derived trees is Lipstchitzian, so that the mapping has the locality property.

Theorem 1 *For every TAG G_{lex}, suppose that f is the map used to decode G_{lex} derivation trees into corresponding G_{lex} derived trees, then, for all pairs of G_{lex} derivation trees u, v, there is always a fixed constant $M > 0$ such that f satisfies the following inequality:*

$$d(f(u), f(v)) \leq M \times d(u, v) \tag{3.1}$$

where $f(u)$ and $f(v)$ are the two derived tree of u and v respectively, and d is the tree edit distance (If a map f satisfies the above inequality, it is called a Lipschitzian map).

Proof: Let M be the maximal number of nodes in an elementary tree in the elementary tree set E of G_{lex}. Since E is finite, and all elementary trees in E are finite, it is always feasible to choose such an M. If the tree editing distance between u and v is k, then there are k operations involving the addition and/or deletion of nodes to transform u to v (and vice versa). Thus, each node in u (or v) is labeled by an elementary tree $t \in E$, that has at most M nodes. Moreover, since an adjunction between two elementary trees does not change the meaning or the number of nodes in them, it is deduced that the number of node differences (using node addition, deletion, or relabeling) between $f(u)$ and $f(v)$, two derived trees of u and v, are at most $M \times k$. Therefore, the tree editing distance between $f(u)$ and $f(v)$ is less than or equal to $M \times d(u, v)$.

Theorem 1 shows that the genotype-to-phenotype map in the TAG-based representation has the locality property: “small change in genotype will result in small change in phenotype”. In fact the scale of change from genotype to phenotype is bounded by the size of the biggest elementary tree in G_{lex}. Having proven that, it is also possible to change this scale of change by crafting the elementary trees of G_{lex} with different sizes. In practice, throughout this thesis, M is usually relatively small.

For Rothlauf’s framework [Rot2002], we discuss the redundancy properties in chapter 8. The other issues are discussed in concrete terms in the next subsection.

3.2.3 A Working Example of TAG-based representation

In this subsection, an example of TAG-based representation and its translation from genotypes to phenotypes are illustrated. The grammars are taken from one of our work [Hao2005]. The task in citehoaihao2 is to elvolve equations defined by the following CFG:

$G = (N, \sum, P, S)$, where N={S,T,OP}, $\sum$={pow,+,-,*,/,p,n,s,t,l,o,co,chla,r1,r2}. The rule set P is as follows:

$S \rightarrow T$

$T \rightarrow T\ OP\ T$

$T \rightarrow T\ pow\ r1$

$T \rightarrow p|n|s|t|l|o|co|chla|r2$

$OP \rightarrow +|-|*|/$

Using algorithm given in appendix A, we have the corresponding LTAG G_{lex} as follows: $G_{lex} = (\sum, N, I, A)$, where $\sum$, N are the same as in G. The elementary tree set $I \cup A$ of G_{lex} is given in figure 3.12.

Figure 3.13 follow demonstrates how a genotype in TAG-based representation for this problem (a derivation tree in G_{lex}) can be translated into its corresponding phenotype (a derivation tree in G).

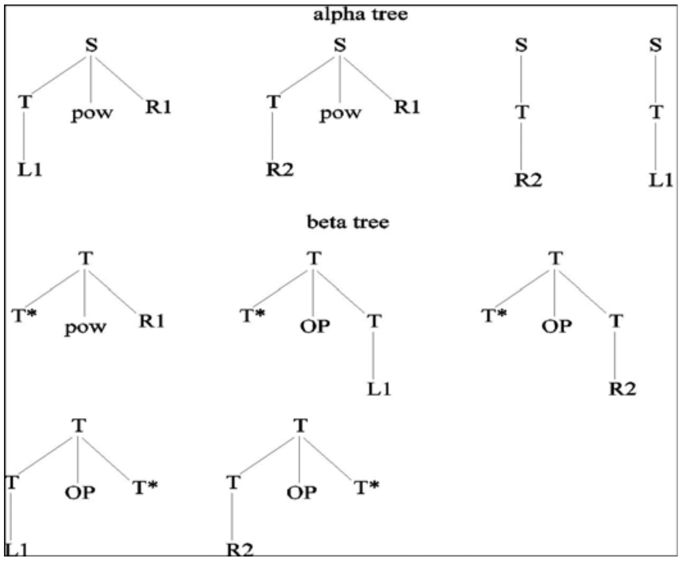


Figure 3.12: The elementary trees for G_{lex}. L1 is a lexicon that can be substituted with any lexeme in (p,n,s,t,l,o,co,chla,r1,r2)

.

3.2.4 Why TAG-based representation ?

In this subsection, the justifications for the use of TAG derivation tree as a new representation for genetic programming are outlined. These discussions will be amplified in more detail in later chapters.

The main justification for the use of TAG-based representation in GP is that it can be exploited to solve a number of difficulties in GP and GGGP metioned in chapter 1 and 2. Of course, it is not the only representation able to provide such solutions, and indeed our primary emphasis is on the importance of these design issues as exemplified by TAG representation. However it is, perhaps, unusual in the range of issues that it is able to handle (as depicted in the sequel chapters).

The non-fixed arity property of TAG derivation tree permits the design of a wide range of genetic and/or search operators, which are difficult to implement in many other representations, notably GP and GGGP. This is discussed in chapters 4 and 6. In particular, it is easy to design operators that make small and bounded changes. In other words, the degree of change is controllable. Together with the locality property of the genotype-phenotype mapping, it means that

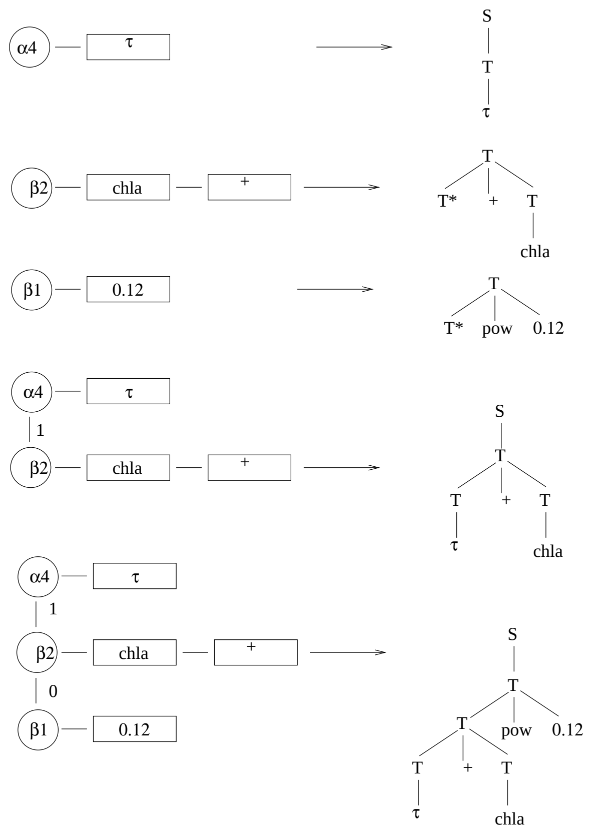


Figure 3.13: Translation from a genotype to a phenotype. The first four parts are the intermediate steps. The final genotype and final phenotype are given at the bottom of the figure

the change induced by those operators is also bounded and controllable in the phenotype space. These properties, in turn, can be used (chapter 8) to soften the structural difficulty problem in GP (and GGGP), which appears to result from the unboundedness (or discontinuity) of standard GP operators. A second advantage of the controllability of the scale of operator change lies in the ability to simply conduct studies of GP fitness landscapes, especially on syntactically constrained domains, since it is possible to make short moves from one point in a search space to others in its neighbourhood; this is otherwise particularly difficult on syntactically constrained domains. The study of fitness landscapes is covered in chapter 9.

Finally, since the TAGs are tree grammars, and their derivation trees are hierarchical object-trees, it is possible to define and investigate the propagation of search schemata, in which the concept of schemata unifies all three aspects of schemata in GGGP, namely as program component, as search space component, and as language formalism. This is discussed further in chapter 7.

3.3 Conclusion

In this chapter, a new representation for GP and GGGP has been introduced, based on a relatively new formalism from the field of natural language processing, namely tree adjoining grammars (TAGs). We pointed out that a number of well-known virtues of TAGs for handling issues in natural language processing might also be helpful in the field of GP (and especially GGGP). The representation has been informally tested against well-known principles for EA representation from the literature. As a foundation chapter for the thesis, it not only outlines some justifications for the use of the new representation, but also points to where those justifications are examined in detail in the thesis.

Chapter 4

A Tree Adjoining Grammar Guided Genetic Programming System (TAG3P)

4.1 Introduction

This chapter describes in detail the major components of a tree adjoining grammar guided genetic programming system (TAG3P) based on TAG-based representation in the previous chapter. Firstly, TAG3P program representation is introduced. Then, it presents an algorithm for initialising a random population for TAG3P and prove its correctness. Next, the fitness evaluation and genetic operators in TAG3P are described. Finally, the parameters for TAG3P are defined. It is assumed that a tree adjoining grammar G_{lex} is used to generate programs. On the top level, TAG3P main evolution cycle is similar to other genetic programming systems as follows:

```
1)  Create a population of individuals, where each individual is
    a $G_{lex}$ derivation tree.  It is called the old population.
2)  All individuals ($G_{lex}$ derivation tree) in the old population
    are translated into $G_{lex}$ derived trees.
    Then, on these derived trees, the fitness of the individuals
    in the old population are calculated.
3)  Select individual(s) from the old population according to
    some fitness-based selection mechanisms.
4)  Apply selected genetic operator(s) on selected individual(s)
    according to the operator probability
    and copy the product(s) to the new population.
5)  Repeat from 2 to 4 until the new population is filled up with
    new individuals of $G_{lex}$ derivation trees.
6)  The old population are removed and
    the new population becomes the old population.
7)  Repeat from 2 to 6 until a number of generations is reached
    or some terminating criteria are met.
8)  Report the best individual (in terms of fitness)
    in the last generation as the solution.
```

4.2 The Components of a Tree Adjoining Grammar Guided Genetic Programming (TAG3P)

As in a standard genetic programming system [Koz92], TAG3P consists of five basic components, namely, program representation, initialisation procedure, genetic operators, fitness evaluation, and parameters. The first component defines how each individual (program) in the population is structured. As indicated in the algorithm in the previous section, each individual in a TAG3P population is a derivation tree of the LTAG (G_{Lex}) used to syntactically constrain the language of the programs for solving the problem being concerned. The second component is an algorithm for creating an initial population of individuals at random. Its main task is to accomplish step 1 in the algorithm in the previous section. Genetic operators, the third component, are a set of transformations from $X^n \rightarrow X$, where X is the space of all feasible individuals and X^n is the n-product space of X. In TAG3P, X is the set of G_{lex} derivation trees and n is 2 for the crossover operator and 1 for the other operators. It is noted that selection mechanisms, the ways that individuals are chosen in step 3, are also considered as a part of this component [Koz92]. The fourth component, which is problem dependent, specifies how the goodness of each individual (program) in the population is measured. The last component is tunable parameters used to adjust various aspects of TAG3P ranging from the bound of the individual size to the likelihood of applying different genetic operators and so on. In the following subsections, each of the five basic components of TAG3P are discussed in more detail.

4.2.1 Program Representation

Similar to GE and GEP described in chapter 2, TAG3P uses a genotype-phenotype map. The special form of derivation tree in LTAG presented in the previous chapter is used as genotype structure for program representation. TAG3P can handle problems with context-sensitive syntactical constraints, context-free syntactical constraints, or (as in standard GP) no syntactical constraints. Therefore, the phenotype can be one of three cases. In the first, an LTAG grammar G_{lex} is used on its own as the formalism for language bias declaration. In that case, the phenotype is the derived tree of G_{lex}. In the second case, the context-free grammar (CFG) G is used to generate the strongly lexicalised LTAG G_{lex}. The derivation tree of G_{lex} is used as the genotype, and the phenotype in that case is the derivation tree of G (derived tree of G_{lex}). In the final case, a set of GP functions and terminals is used to create a context-free grammar G in the manner described in [Whi1996] (page 130). It was proven in [Whi1996] that there is a one-to-one correspondence between the derivation trees of G and the expression trees in GP. Based on this, Whigham concluded that standard GP is just a special case of GGGP. The mapping process can be summarised in Figure 4.1 follows, where the second phase of the map is optional.

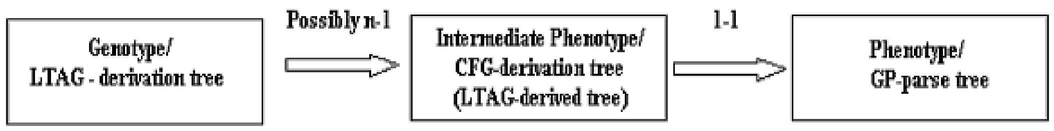


Figure 4.1: Mapping Process

A subcode in each individual chromosome in TAG3P is a subtree. Every subcode is expressed (i.e. it is used to code for the phenotype). However, unlike other GP and GGGP systems, in TAG3P, it is possible to measure how each subcode contributes to the fitness of the individual. Thanks to the non-fixed arity properly described in the previous chapter, it is possible to temporarily remove any subcode in an individual and leave the individual still valid (fitness calculable). Therefore, one way to determine how each subcode contributes to the fitness of an individual is accomplished by comparing the fitness of the individual with and without the presence of the subcode. This is the usual way that a genetic engineer determines the effect of a gene in genetic engineering [Drl1984]. The possibility of measuring the contribution of each subcode to the fitness of the individual can be useful in a number of ways. Firstly, it helps TAG3P to pinpoint which subcode is not contributing to the fitness of the individual; it could be eliminated to simplify the individual. This might be done by checking if the fitnesses of the individual, before and after the subcode is temporarily removed are the same. We note that the way the individual scores its fitness on individual fitness case might change, so long as the overall fitness does not change. For instance, if the fitness of the individual is measured by summing up its scores on 10 fitness cases, then a subcode is called non-fitness contributing in this individual if the sum score on 10 fitness cases of the individual does not change when the subcode is removed, though the individual's fitness score on some of the 10 fitness cases might change. In the next chapter, we will show how this can be used to simplify the solutions when they are found, which is usually complex and somewhat cumbersome problem in GP (GGGP). Furthermore, it is also possible to simplify the solution (or any individual) by sacrificing some of its fitness quality. To do this, one can detect and eliminate any subcode within this individual such that, when removing it, the amount of change in the individual fitness does not exceeds a predefined bound. This might be potentially useful in the application of genetic programming to machine learning as in [Fre2002], as it becomes possible to trade off the fitness (accuracy) of the evolved decision tree against its complexity (generalisation). However, this is not studied in this thesis. In similar manner, one can determine which subcode contributes most to the fitness of the individual, and which makes the individual's fitness bad. By looking at those subcodes, as in the next chapter, some useful information about the evolutionary process can be extracted. Lastly, it is possible in TAG3P to turn on and turn off any subcode at will. Although it is not yet shown in this thesis, this property might be potentially useful when one wants to set a bias controlling which part of the chromosome (program) will undergo changes by genetic operators.

4.2.2 Initialisation Procedure

The second component in TAG3P is an algorithm for creating an initial random population of individual (G_{lex} derivation trees). It is a repetition of the process of generating one individual (one G_{lex} derivation tree) at random. The process to generate a G_{lex} derivation tree at random starts with choosing a random size in a predefined range of integer numbers. Then, it proceeds by randomly picking an α-tree from the elementary tree set in G_{lex} to make an initial G_{lex} derivation tree. This derivation tree is subsequently extended with β-trees drawn at random from the elementary tree set in G_{lex} by using adjunction at random places. This process finishes when the size randomly chosen above is reached. The initialisation procedure for TAG3P is formalised as the following algorithm

```
1)   FOR $i = 1$ TO POPSIZE DO
2)   Choose a a random size $l$ between MINSIZE and MAXSIZE.
3)     Pick an $\alpha$-tree $\alpha_1$ at random and set tree $T = \alpha_1$.
4)     FOR $j = 1$ TO $l - 1$ DO
5)     Set $V$={ node $n$ in $T$ such that $n$ has at least
         one NULL-adjoining address}
6)     Pick a node $n$ in $V$ in an uniformly random manner.
7)     Randomly pick a NULL-adjoining address $a$ in elementary tree $n$.
8)     Among all $\beta$-trees in the elementary tree of $G_{lex}$
         that can adjoin to $a$, choose a tree $t$.
9)     Adjoin $t$ to $a$ in $T$ and update $T$.
10)    ENDFOR
11)  Set individual $i$-th as $T$
12)  ENDFOR
```

where MINSIZE and MAXSIZE are adjustable parameters for designating the range of individual size and will be described in section 4.2.6; a NULL-adjoining address means at this adjoining address there is no elementary tree being adjoined at it yet. In order to be a genuine algorithm, it is strictly required that the above process always stop and give the desired results. However, it is not obvious that this process will finish and give a population of only valid G_{lex} derivation trees. Therefore, a theorem is given and proven here.

Theorem 2 *Given that every α-tree in G_{lex} can be adjoined by at least one β tree, the initialisation procedure above always finishes and gives the desired results.*

Proof: To prove the theorem, it is sufficient to prove that for all steps from 3 to 8, the procedure always produces desired results.

For step 3, since the set of α-trees in G_{lex} is not empty, it is always possible to choose an α-tree α_1 from it. If l is 1, the process stops after setting $T = \alpha_1$ in step 3. On the other hand, if $l \geq 2$ the above process will extend the tree with some β-trees since, according to the hypothesis in the theorem, it is always possible to adjoin some β-trees to α_1. Consequently, at all times, T has some leaf nodes that are β-trees. As a leaf node is defined in the previous chapter as a node that has no adjunction to it, it guarantees that $V \neq \emptyset$ in step 5. Consequently, it is always possible to choose n in V as in step 6. According to the definition of β-trees, there is always at least one address that can be adjoined by a β tree in the elementary tree set of G_{lex} (the trivial case is that the root or the foot of the tree can be adjoined by itself). Therefore, it is always possible to choose the address a in 7. As defined in the previous chapter, for every adjoining address, there is at least one β-tree that can adjoin to that address (according to the definition, an adjoining address in an elementary tree is a node that has the same label as at least one β-tree). Thus, it is always possible to choose t in 8. Therefore, all steps from 2 to 8 always finish and produce meaningful and desired results. That completes the proof.

The proof of theorem 2 also shows that the time complexity for randomly creating an individual (G_{lex}) is quadratic in its size, since before each node is added, the tree is searched to find all possible adjoining address for the new node. It is stated as an corollary as follows:

Corollary: *The time complexity of the algorithm for initialising an indivdual in TAG3P is $O(l^2)$, where l is the size of the individual. Therefore, the time complexity of the initialising procedure is $M \times l^2$, where M is the population size.*

We note that an algorithm, in which adjoining addressed are cached while the individual is being built, could potentially reduce the initialisation times cost to linear, but also requiring more memory.

The condition of the theorem is not a hard constraint. In general, G_{lex} should always satisfy it, since if there is any α-tree in the elementary tree set of G_{lex} that cannot be modified by adjoining with some β-trees then we can simply discard such α trees to treat them separately. The number of such trees are finite because it cannot exceed the number of elementary trees in G_{lex}.

Theorem 2 shows that given any size, TAG3P can produce randomly an individual with this exact size. This unique size control property allows TAG3P to have an absolutely uniform initialisation according to size, which is difficult for other genetic programming systems. As pointed out in [Rat2000], the initialisation process should not be underestimated in grammar guided genetic programming. In one of the author's work [Hao2004], it has been shown that using the above initialisation procedure and exporting the population to a GGGP system instead of using the standard initialisation procedure as in [Whi1996] helps GGGP perform better on a number of standard problems.

4.2.3 Fitness Evaluation

To evaluate the fitness of an individual, it is first translated into an derived tree in G_{lex} (derivation tree of G if G is used). Then, the evaluation is processed on that derived tree.

In the next two subsections, the third component of TAG3P - the genetic operators- will be described.

4.2.4 Main Genetic Operators

This subsection discusses the main genetic operators in TAG3P, namely, selection mechanisms, reproduction, subtree crossover, and subtree mutation. Some more innovative operators are described in the next subsection.

Selection Mechanisms

In TAG3P, all common selection mechanisms from other evolutionary algorithms can be used. In particular, the two most common selection mechanisms are fitness proportionate and tournament selection as described in chapter 2.

Reproduction

Reproduction in TAG3P is just like other genetic programming systems, wherein a proportion of the population is chosen based on fitness and then copied to the new population.

Crossover

To do crossover, firstly, two parent individuals t_1 and t_2 are chosen from the population by a selection mechanism. Then, the crossover between these is applied. It is done, by first choosing two nodes in the two trees that are compatible. Two nodes in the two trees are compatible if the subtrees under each of them can adjoin to the parent node of the other in each other's tree. The above process of choosing two compatible nodes is repeated until either two such nodes are found in t_1 and t_2, or the number of trials exceeds a predefined bound. In the second case, crossover is aborted. If the two compatible nodes are found, the crossover is completed by swapping the two subtrees underneath these two nodes. In addition, there might be some overhead checking that the individuals produced by crossover satisfy some conditions such as size constraints (e.g. if the size of the produced individuals must stay within a predefined range between MINSIZE and MAXSIZE). The crossover operator process is summarised as follows:

```
1)  Randomly choose a node $n$ in $t_1$, except the root.
    Suppose that in $t_1$, $n$ is currently adjoined to node $h$
    at address $a_1$.
2)  Randomly choose a node $m$ in $t_2$, except the root.
    Suppose that in $t_2$, $m$ is currently adjoined to node $k$
    at address $a_2$.
3)  IF $\{$ elementary tree $n$ can be adjoined into elementary tree $k$
    at address $a_2$ $\}$
    AND $\{$ $m$ can be adjoined into elementary tree $h$
    at address $a_1$ $\}$
    THEN
     The two subtrees underneath $n$ and $m$ in $t_1$ and $t_2$ are swapped.
     Goto 5.
4)  LOOP from 1 to 3 for a maximal MAXATTEMPT times.
5)  Set two resultant trees as children of $t_1$ and $t_2$.
```

Figure 4.2 illustrates how the crossover operator works.

There are some possible variants on the implementation of the crossover operator. In the above procedure, the crossover operator is implemented in adjoining-context preserving manner. In other words, the adjoining addresses a_1 and a_2 are still adjoining addresses for m and n after the crossover. It is possible to implement crossover in non-adjoining-context preserving manner. This form allow m and/or n to change the adjoining address after the crossover. This can be done if step 3 is changed as follows:

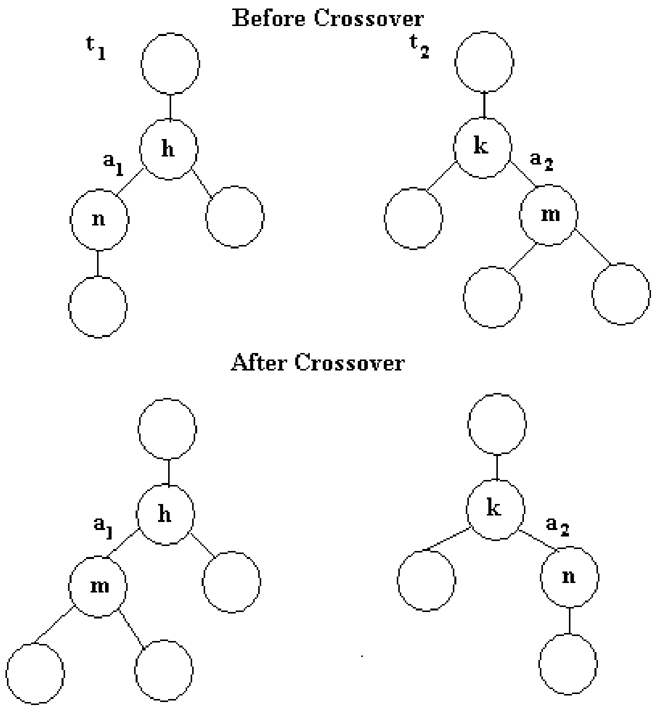


Figure 4.2: Crossover in TAG3P

```
3)  IF { elementary tree $n$ can be adjoined into elementary tree $k$
    at an address $ad_1$, where $ad_1$ is not currently adjoined }
    AND {$m$ can be adjoined into elementary tree $h$
    at address $ad_2$, where $ad_1$ is not currently adjoined }
    THEN
     The two subtrees underneath $n$ and $m$ in $t_1$ and $t_2$ are swapped.
     Goto 5.
```

Moreover, unlike standard GGGP, it is easy to add size constraints on the individuals produced by the crossover operator in TAG3P. For instance, in order to keep the size of the produced individuals always within a predefined range of sizes, a checking procedure can be added at the end of step 3. The case that the size of any produced individual exceeds the predefined range between (two integer numbers) MINSIZE and MAXSIZE can be handled in two ways. The first is to reduce/increase the size of the individual by randomly deleting/inserting some nodes in it. The other is to simply discard the produced individuals and restart the crossover process from the beginning, provided that the number of trials has not exceeded a predefined bound. A Boolean predicate called *_cut* is used to control whether the first or the second ways should be used. The following pseudo code summarises the implementation of this type of size constraint.

```
IF the sizes of the two resultant trees
 are between MINSIZE and MAXSIZE
Goto 5.
ELSE
IF predicate _cut is set THEN
   IF the size of resultant tree(s) exceeds MAXSIZE
    Randomly trim the subtree under $n$ (or $m$) using
    deletion operator described in the next subsection until
    the size of the resultant tree(s) is MAXSIZE.
   IF the size of resultant tree(s) is below MAXSIZE
    Randomly add to the subtree under $n$ (or $m$) using
    Insertion operator described in the next subsection until
    the size of the resultant tree(s) is MINSIZE.
```

Another instance of size constraints is size-fair constraint. The size-fair form of crossover requires the size of the two subtree to be equal. As shown in [Lan2000b], the size-fair crossover can help tree-based genetic programming systems to reduce the code-bloat effect (i.e, the size of trees in the population expands quickly). The code-bloat effect also appears in grammar guided genetic programming [Whi1996] (pages 54-55). However, since the rule-based nature of derivation tree in CFGs, it is difficult to implement size-fair constraints in GGGP. On the other hand, the LTAG-derivation tree is an object-based tree. It is easy to define size-fair crossover. In order to implement it, step 3 above is amended as follow:

```
3)  IF { elementary tree $n$ can be adjoined into elementary tree $k$
    at address $a_2$ }
    AND { $m$ can be adjoined into elementary tree $h$
    at address $a_1$ }
    AND { the difference in size between the subtrees
    under $m$ and $n$ is less than or equal to $\delta$ }
    THEN
     The two subtrees underneath $n$ and $m$ in $t_1$ and $t_2$ are swapped.
     Goto 5
```

δ above is a predefined positive integer to limit the maximal difference allowed in size-fair crossover. When δ is 0, the crossover is called absolutely size-fair.

Subtree Mutation

In sub-tree mutation, a randomly chosen subtree (subcode) is removed and replaced with a newly generated subtree (subcode), which is about the same size. The new subtree is generated in a similar way in step 3 to 9 in the initialisation procedure in the previous section. The procedure for mutation of a tree (program) t is as follows:

```
1)  Randomly choose a node $n$ in $t$.  Suppose that $n$ is
    currently adjoined to node $h$ at address $a$ in $t$.
    The size of $t$ is $l$.
2)  Calculate the size $l_1$ of the subtree under $n$.
3)  Remove the subtree under n.
4)  Randomly choose a positive $l_2$ such that
    $|l1 - l2| \leq \delta$ and $l - l_1 + l_2$ is between MINSIZE and MAXSIZE.
5)  Randomly choose an elementary tree $e$ that can be
    adjoined to $h$ at $a$.
6)  Set tree $T = e$.
7)  Randomly grow $T$ with size $l_{l2}$ in the same manner as
    Steps 3 to 9 in the population initialisation procedure above.
8)  Adjoin (connect) $T$ in $h$ at $a$.
```

δ is the maximal difference allowed between the removed subtree and the newly generated subtree. Figure 4.3 depicts how the subtree mutation works in TAG3P.

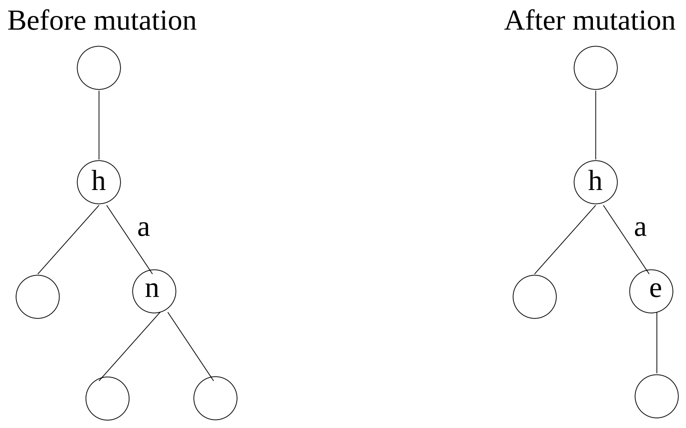


Figure 4.3: Subtree mutation in TAG3P

From theorem 2, it is obvious that the subtree mutation procedure always finishes (in time quadratic in the size of the tree) and gives desired results.

As with subtree crossover, it is possible to implement subtree mutation in adjunction-context-preserving (in the above procedure) and non-adjunction-context-preserving manners. For non-adjunction-context preserving subtree mutation, the above procedure is modified accordingly as follows:

```
     ....
4)   Randomly choose an currently adjunction-free address $ad$ in $h$.
4b)  Randomly choose an elementary tree $e$ that can adjoined to $h$
     at $ad$.
     ....
7)   Adjoin (connect) $T$ in $h$ at $ad$.
```

4.2.5 Other Operators

Thanks to the non-fixed arity property in TAG-based representation, a number of search operators can be designed. They can ben seen either as general-purpose local search operators or as asexual genetic operators. Some of them are bio-inspired. They can be divided into two groups. The first group of operators operates on the node level, whereas the second operate within the node level. In other word, the first group of operators deals with adjunctions in an individual (a G_{lex} derivation tree), whereas the second deals with substitutions. The first group consists of insertion, deletion, node replacement, replication, duplication, and truncation. The second group includes single-lexeme mutation, node-lexeme mutation, and global lexeme mutation. The first group is only mentioned briefly here as they will be described in more details in chapter 6 and 8.

Insertion

If the size of the individual is less than MAXSIZE then insertion simply adds a leaf to the individual. The insertion operator is described in more detail in chapter 6, where it is coupled with deletion as a dual operator.

Deletion

If the size of the individual is more than MINSIZE then deletion simply removes a leaf of the individual. It is easy to see that deletion is dual to insertion. The deletion operator is described in more detail in chapter 6, where it is coupled with insertion.

Node Replacement

Node replacement has been studied elsewhere in GP [MKa95, Lan2002]. Node replacement resembles the bit-flip in traditional GA, where the change of content is minimal (i.e the change is at one node of the tree only). However, it is rather difficult in GGGP to design a similar operator. With TAG-based representation, such an operator is readily implemented. The procedure for node replacement on an individual, in TAG3P, is as follows.

```
1)  Randomly choose a node $n$ in $t$.
    Suppose that $n$ is adjoined to node $h$ in $t$ at address $a$.
    Suppose that there are $p$ nodes $n_1$, $n_2$,...,$n_p$ are adjoined to n
    in $t$ at addresses $a_1$, $a_2$,...,$a_p$.
2)  Choose a node $m$ at random such that $m$ can adjoin to $h$ at $a$
    and $m$ has (at least) p adjoining addresses that has the same
    labels with $a_1$, $a_2$,...,$a_p$.
3)  Replace $n$ with $m$.
```

Figure 4.4 shows how node replacement works.

Node replacement operators can also be implemented in a biased way. Since the elementary tree set is predefined and finite, it is possible to craft a predefined set of replacement elementary tree for each elementary tree in G_{lex}. Thus, when node n is chosen for replacement in step 1 of the procedure above, node m will be one of the elementary trees in this set. Unbiased node replacement is used with insertion and deletion as prime operators for studying fitness landscape on syntactically constrained domains in chapter 9.

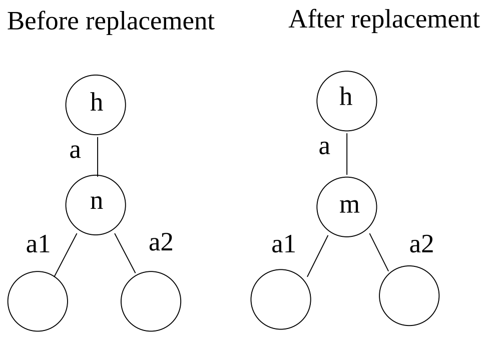


Figure 4.4: Node replacement in TAG3P

Relocation

In relocation, a random subtree (subcode) is disconnected from an individual (a G_{lex} derivation tree). This subcode is then randomly adjoined at other place in the individual. It is noted that relocation neither changes the size of the individual nor introduces any new genetic material. Relocation is described in more detail in chapter 6.

Duplication

In duplication, a random subtree (subcode) is copied and is randomly adjoined to other place in the chromosome tree. Duplication operator is discussed and studied in more detail in chapter 6.

Truncation

In truncation, a random subtree (subcode) in an individual t is chosen. Then, this subtree is removed from t if the the size of the resultant tree is not less than MINSIZE. The procedure for the truncation operator is as follows.

```
1)  Randomly choose a node $n$ in $t$.
    Suppose that size of $t$ is $l$ and size of the subtree under $n$
    is $l_n$.
2)  IF $l - l_n \leq$ MINSIZE THEN
     Remove the subtree underneath $n$ from $t$
     Goto 4.
3)  Loop from 1 to 2 for maximal MAXATTEMPT times.
4)  Update and return $t$.
```

Figure 4.5 shows how truncation works.

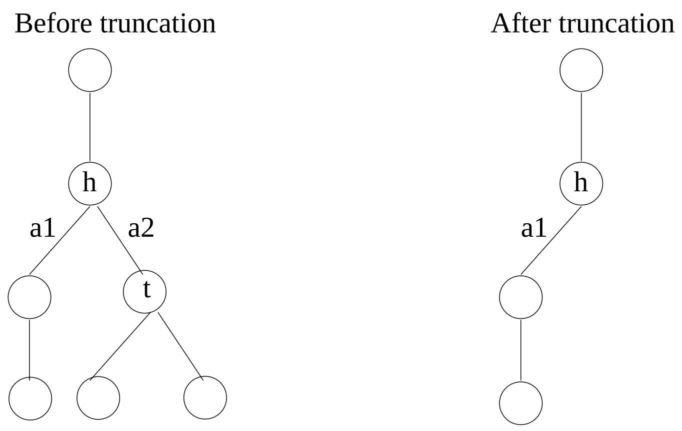


Figure 4.5: Truncation in TAG3P

Lexemes Mutation Operators

The second group of unary operators is lexeme mutations. As described in the previous chapter, for the special form of LTAG derivation tree used for TAG3P, substitution becomes an in-node operation. Each node in an individual is attached by a list of lexemes, used for substitutions in the tree, represented at that node. Therefore, there are three possible levels for implementing lexeme mutation in TAG3P. The difference among these levels is the scale of change it brings to the lexemes used in an individual, ranging from one lexeme to all lexemes attached to one node, and to all lexemes in an individual. In the first and lowest level, node-based single lexeme mutation, a node in the chromosome tree is chosen at random. Then, one randomly chosen lexeme in the lexeme list attached to that node will be replaced by another random lexeme from the same vocabulary. In the second level, node based lexeme mutation, after a node is chosen at random, the whole lexeme list attached to that node is mutated using node-based single lexeme mutation. In the last level, all lexeme lists attached to all nodes in the chromosome tree are mutated using node based lexeme mutation. The lexeme mutation can also be implemented in a deterministic manner or by using local search algorithms. Because of the lexicalisation in LTAG-based representation, it is possible to efficiently implement a mixed strategy search, where the structure is evolved by genetic search and lexicons are discovered by other heuristics or by traditional deterministic search strategies.

4.2.6 Parameters

There are a number of user-control parameters in TAG3P. Their meanings are given below.

POPSIZE - the population size.

MAXGEN - the maximal number of generations.

MAXSIZE - the maximal size (maximum number of nodes) of chromosome trees.

MINSIZE - the minimal size (maximum number of nodes) of chromosome trees. It must be at least 2.

MAXATTEMPT - the maximal number of attempts in each genetic operator.

OPERATORPROBS - the probabilities for applying operators (crossover, mutation, insertion, deletion,...).

δ - the maximal difference allowed in size between new subcode and old subcode when the operators (crossover, mutation,...) are implemented in size-fair fashion.

_cut - a predicate that can be set as either ON or OFF. It is used in the crossover operator to trim or fill the resultant trees, when their sizes go out of bounds (MINSIZE and MAXSIZE).

4.3 Some Information on TAG3P Implementation

TAG3P was implemented in C/C++. The platform used to run TAG3P for all experiment in this thesis is Sun Workstation in a cluster marchine, with Linux as the operating system, at Australian Defence Force Academy. The code is rather compact and run fast (typically some minutes for problems with population size 500, number of generations 50, number of fitness cases 20). The code is available for download at:
www.cs.adfa.edu.au/we/hoai.htm.

4.4 Conclusion

In this chapter, the structure of a tree adjoining grammar guided genetic programming (TAG3P) system, built on the TAG-based representation in the previous chapter, has been laid out. Some useful properties of the TAG-based representation given in the previous chapter has been transferred into the design of TAG3P. In particular, the non-fixed arity property in the TAG-based representation is very useful for implementing components of TAG3P. Firstly, it makes possible the measurement of how the presence of each subcode affects the individual fitness. That, in turn, can be used to simplify solutions (individual) at the end of (or during) the evolutionary process. Some examples of this is given in the next chapter. Secondly, the non-fixed arity property makes initialisation in TAG3P much easier and more uniform than in CFG-GP. Lastly, it facilitates the design of a wide range of general-purpose search operators for syntactically-constrained domains. Some of them are bio-motivated and can potentially be used either as genetic operators or as local search operators. Moreover, those operators can be used to make different degree of changes on an individual ranging from a tiny change within one node (node-based single lexeme mutation) to the whole individual itself (e.g. subtree mutation when the node chosen to be mutated is the root node of the individual). More importantly, the degree of change on an individual induced by some operators (such as insertion, deletion, node replacement) is bounded. Therefore, it is possible to control the degree of change on each individual when repeatedly using those operators. The usefulness of some TAG3P operators, and the controllability of the amount of change brought by them on an individual, is shown in subsequent chapters.

Chapter 5

TAG3P: Preliminary Comparison

In this chapter, the robustness of the genetic programming system (TAG3P) designed in the previous chapter is tested on a number of standard problems. The results are compared with standard genetic programming (GP) and Whigham's grammar guided genetic programming system (CFG-GP). Some of the results have been published in [NXH2001a], [NXH2001b], [NXH2002a], [NXH2002b], [NXH2002c], and [NXH2002d]. This chapter is structured as follows. Firstly, all the problems are stated. Then, the experiment setup is described. Next, the results are given and discussed. Finally, using some of the rather unique properties of TAG3P described in the previous chapter, we conduct some further analyses on TAG3P runs to discover useful information about the dynamics of its evolutionary process.

5.1 Test Problems

To test the performance of TAG3P compared to standard GP and CFG-GP, eight problem instances of four standard problems from [Koz92] are employed as test suite. The four problems are simple symbolic regression, 6-multiplexer, symbolic integration, and symbolic differentiation. The description of each problem and problem instance is briefly stated in the following subsections. It is noted that a full comparison would compare the performance of the systems over a range of parameter settings, but that because of computational limitations, only a sample of values is used in the thesis.

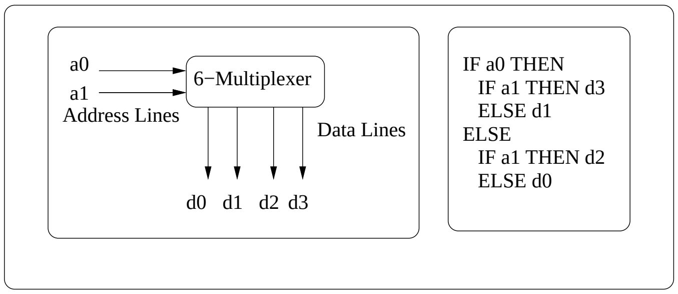
5.1.1 Simple Symbolic Regression Problem

In the simple symbolic regression problem, the task is to learn a function of one independent variable in symbolic form such that it fits a given finite sample of data. As in [Koz92], the function and terminal set are $F = \{+, -, *, /, sin, cos, ep, rlog\}$ and $T = \{X\}$. The function set can be divided into three groups. The first group contains arithmetic operators: addition, subtraction, multiplication, and division. The division operator is protected in the sense that if the denominator is 0, it will return 1. The second group has two basic trigonometric functions sine and cosine. The third group has the two transcendent functions, namely, the exponential function and logarithm function. The logarithm function is protected by taking logarithm of the absolute value of the input unless the input value is 0, in which it returns 0. The number of sampled data points is 20 and the sampling interval is $[-1..1]$.

In [Koz92], the target function for genetic programming is the quadtic polynomial function $X^4 + X^3 + X^2 + X$. In this chapter, a family of four target functions in four problem instances are used.
$F1 = X^3 + X^2 + X$
$F2 = X^4 + X^3 + X^2 + X$
$F3 = X^5 + X^4 + X^3 + X^2 + X$
$F4 = X^6 + X^5 + X^4 + X^3 + X^2 + X$
It is noted that $F1 - F4$ are a family of polynomial functions of increasing order of structural complexity, where $F_i = F_{i-1} * X + X$ with $i = 2, 3, 4$. The purpose of using theses four polynomial target functions of increasing order of structural complexity is not only to test the robustness of TAG3P against standard GP and CFG-GP but also to investigate how it copes with a scaling in structural complexity of the target function.

5.1.2 6-Multiplexer Problem

A 6-multiplexer is a Boolean device that has six inputs consisting of two address lines and four data lines. To produce the output, a 6-multiplexer uses the two address lines as a two binary digit number in order to select one of the four possible data lines. The task is to learn the Boolean 6-multiplexer function from all 64 possible examples. The function and terminal sets used here are the same as in [Koz92]: $F = \{if, and, or, not\}$ and $T = \{a_0, a_1, d_0, d_1, d_2, d_3\}$. Figure 5.1 shows the structure of a 6-multiplexer and one possible solution represented as a series of if-then-else.



```
IF a0 THEN
  IF a1 THEN d3
  ELSE d1
ELSE
  IF a1 THEN d2
  ELSE d0
```

Figure 5.1: A 6-multiplexer and a solution

5.1.3 Symbolic Integration Problem

The problem of symbolic integration involves finding a mathematical expression that is the integral, in symbolic form, of a given curve. The function and terminal sets used in [Koz92] are the same as those for the simple symbolic regression problem and are used here.

The two problem instances experimented in this chapter are $cos\ x+2x+1$ and $4x^3+3x^2+2x+1$, of which the target symbolic integral functions are $sin\ x+x^2+x$ and $x^4+x^3+x^2+x$ respectively. The number of sampled data points are 50 and the sampling interval is $[0..2\pi]$ in the first case and $[0..1]$ in the second.

5.1.4 Symbolic Differentiation Problem

Symbolic differentiation involves finding a mathematical expression that is the derivative, in symbolic form, of a given curve. The function and terminal sets are the same as in the simple regression and symbolic integration problems ([Koz92]). The given curve is $sin\ x + x^2 + x$, of which the target derivative function is $cos\ x + 2x + 1$. The number of randomly sampled data points is 200 (as numeric differentiation is considered to be much less accurate than symbolic integration); and the sampling interval is [0..1].

5.2 Experiment Setup

All three systems: standard genetic programming, CFG-GP, and TAG3P were tried on eight problem instances of the four standard problems above. The GP and CFG-GP systems are implemented faithfully to the descriptions in [Koz92] and [Whi1996]. All three systems used the same random generators and were run on the same platform. Parameter settings for all three systems were as uniform as possible. They are listed as follows: population size - POPSIZE=500, maximal number of generations - MAXGEN=51, crossover probability=0.9, mutation probability = 0.1, selection mechanism = tournament selection, tournament size of selection = 3, maximal size for TAG3P - MAXSIZE=40; the size in TAG3P is the number of tree nodes; maximal depth for GP and CFG-GP - MAXDEPTH = 15. The initialisation method used for GP was ramped-half-and-half. The grammars (used for CFG-GP and TAG3P) for all problems are listed as follows:

Grammars for the symbolic regression, symbolic integration, and symbolic differentiation problems. The grammars (G and G_{lex}) used for all three problems were the same.

$G = \{\sum, N, P, EXP\}$,

where $\sum = \{+, -, *, /, sin, cos, ep, rlog, X\}$, $N = \{S, PRE, OP, VAR\}$, and the rule set P as follows.

$S \rightarrow EXP\ OP\ EXP$

$S \rightarrow PRE\ EXP$

$S \rightarrow VAR$

$OP \rightarrow +|-|*|/$

$PRE \rightarrow sin|cos|ep|rlog$

$VAR \rightarrow X$

where / is the protected division (return 1 when the denominator is 0), *ep* is the exponential function, and rlog is the protected logarithm function (return the logarithm of the absolute value of the input unless the input is 0, in which case it return 0). $G_{lex} = \{\sum, N, I, A, EXP\}$, where $\sum$ and N are the same as in G and the elementary tree set $E = A \cup I$ is depicted in Figure 5.2.

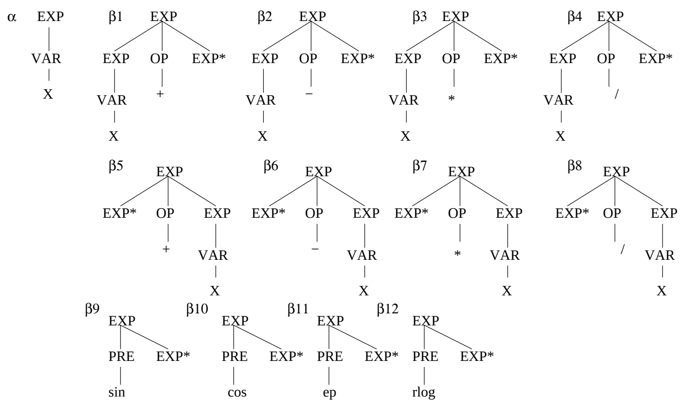


Figure 5.2: TAG elementary trees for symbolic regression, symbolic integration, and symbolic differentiation problems

Grammars for the 6-multiplexer problem. $G = \{\sum, N, P, B\}$, where $\sum = \{a_0, a_1, d_0, d_1, d_2, d_3\}$, $N = \{B\}$, and the rule set P is as follows:

$B \rightarrow if\ B\ B\ B$

$B \rightarrow B\ and\ B$

$B \rightarrow B\ or\ B$

$B \rightarrow not\ B$

$B \rightarrow a_0|a_1|d_0|d_1|d_2|d_3$

$G_{lex} = \{\sum, N, I, A, B\}$, where $\sum$ and N are the same as in G, the elementary tree set $E = A \cup I$ is depicted in Figure 5.3, where TL stands for a lexicon that can be substituted with any of $a_0, a_1,...,d_3$.

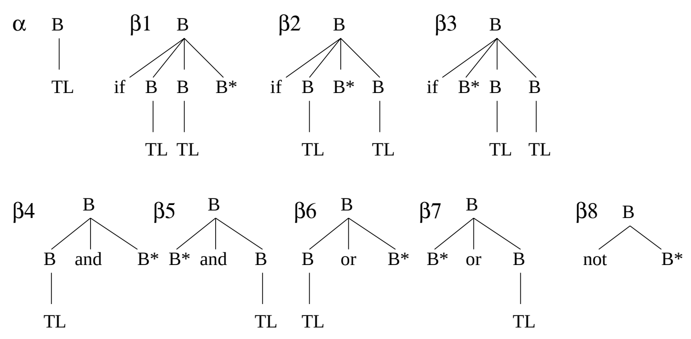


Figure 5.3: TAG elementary trees for the 6-multiplexer problem

The tableaus for symbolic regression, symbolic integration, symbolic differentiation, and 6-multiplexer are as follows.

Table 5.1. Tableau for symbolic regression, integration, and differentiation.

Objective	Find a solution for either symbolic regression, symbolic integration, or symbolic differentiation.
Terminal set	X
Function set	+,-,*,/,$sin, cos, ep, rlog$.
Fitness cases	Random samples of 20 values (symbolic regression); 50 values (symbolic differentiation); 200 values (for symbolic integration) from the intervals of interest.
Raw fitness	Sum of absolute error for all fitness cases.
Standardized fitness	The same as raw fitness.
Hits	Number of absolute errors that is smaller then 0.01.
General Parameter	POPSIZE=500, MAXGEN=51, Tournament selection with size 3.
Success Predicate	A program hits of 20 (symbolic regression); 50 (symbolic integration); 200 (symbolic differentiation)

Table 5.2. Tableau for 6-multiplexer.

Objective	Find a Boolean function whose output is the same as the 6-multiplexer function.
Terminal set	$a_0, a_1, d_0, d_1, d_2, d_3$
Function set	$if, and, or, not.$
Fitness cases	The 64 possible combinations of 6 the inputs;
Raw fitness	Number of fitness cases for which the output is correct
Standardized fitness	Number of incorrect outputs over all fitness cases.
Hits	Equivalent to raw fitness.
General Parameters	POPSIZE=500, MAXGEN=51, Tournament selection with size 3.
Success Predicate	A program scores 64 hits.

5.3 Results and Discussion

For each problem instance, 100 runs were allocated for each system, which made the total number of runs for all three systems (GP, CFG-GP, TAG3P) to be 2400 runs. The following Table 5.3 summarises the proportion of success of all three systems on the eight problem instances tried, where $SYMF_i$ (with i=1,2,3,4) stands for symbolic regression problem with F_i as the target function; $6MUL$ stands for 6-multiplexer problem; $SYMDIFF$ indicates the symbolic integration problem; $SYMITEGCOSX$ and $SYMITEGX3$ are the abbreviations of symbolic integration problem with the curves $cos\ x+2x+1$ and $4x^3+3x^2+2x+1$ respectively.

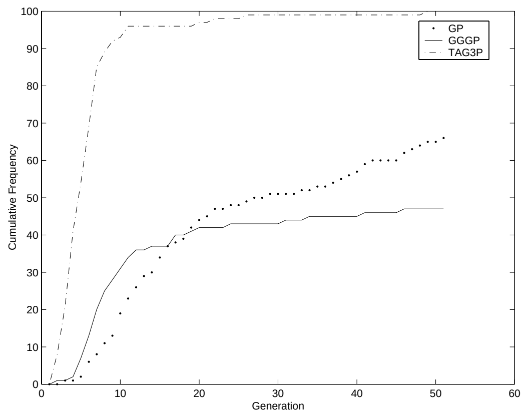
Table 5.3. Proportion of success on eight problem instances.

Problem	GP	CFG-GP	TAG3P
$SYMF_1$	66%	47%	100%
$SYMF_2$	9%	27%	93%
$SYMF_3$	3%	12%	82%
$SYMF_4$	1%	8%	43%
$6MUL$	63%	61%	63%
$SYMITEGCOSX$	81%	83%	100%
$SYMITEGX3$	50%	63%	99%
$SYMDIFF$	70%	77%	87%

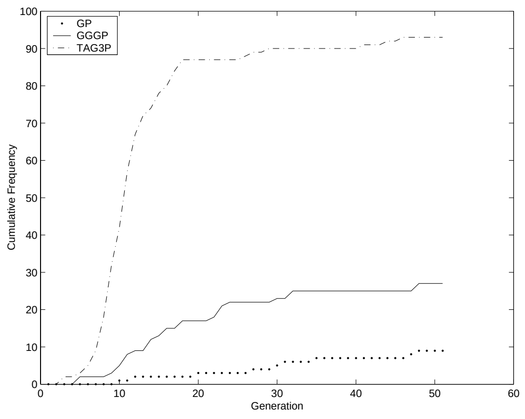
Also, Figures 5.4, 5.5, 5.6, and 5.7 depict the cumulative frequencies of success of GP, CFG-GP (GGGP), and TAG3P on all the problem instances.

The results show that TAG3P is very competitive with GP and CFG-GP. On all problem instances tried, TAG3P performed significantly better than CFG-GP and GP (with statistical confidence level $\alpha = 0.01$ using a one-tailed statistical test of the difference between two binomial variables), except for 6-multiplexer, where all the performances of the three systems are rather similar. In particular, on the symbolic regression problem, TAG3P not only performed substantially better than GP and CFG-GP but also worked well when the structural complexity of the target function was scaled up.

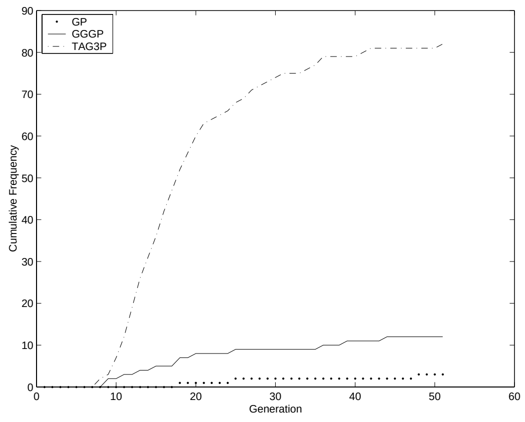
We note that, apart from the fact that these three systems use very different representation and genetic operators, they have different types of complexity bounds on individual programs in their population. For GP, the bound is the depth of the expression tree; for CFG-GP, the bound is the depth of the derivation trees generated by G; and, for TAG3P, the bound is the size of G_{lex} derivation trees. This usually leads to different absolute types and sizes of bounds and thus it is very difficult to tune the different bounds so that the complexity bounds on individual programs in all systems are the same. Therefore, it is possible that the performance differences between the different GP systems might be caused by the different representations (operators) and/or the types and values of the



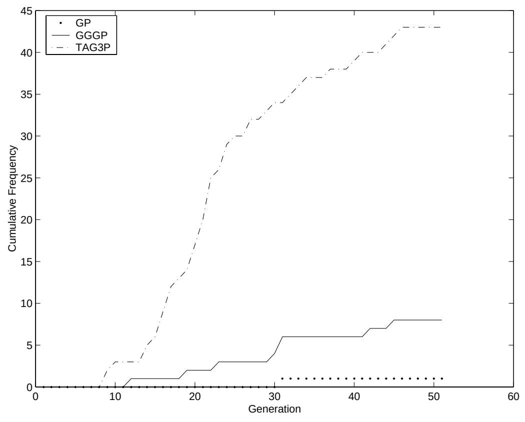
F_1



F_2



F_3



F_4

Figure 5.4: Cumulative Frequencies for Symbolic Regression Problem

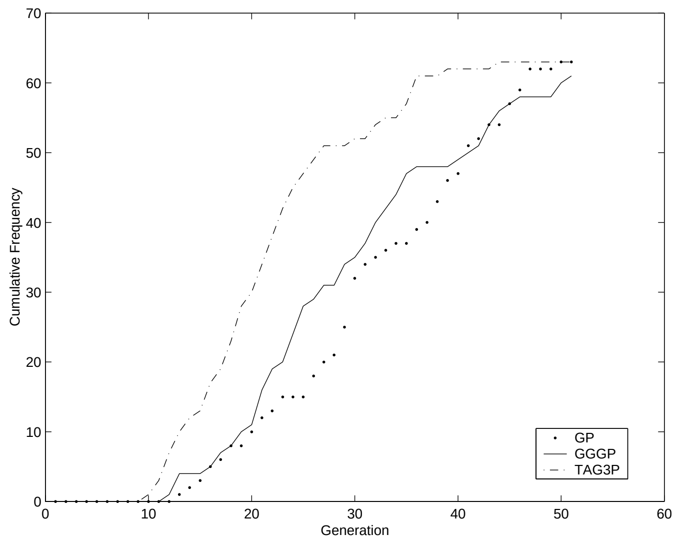
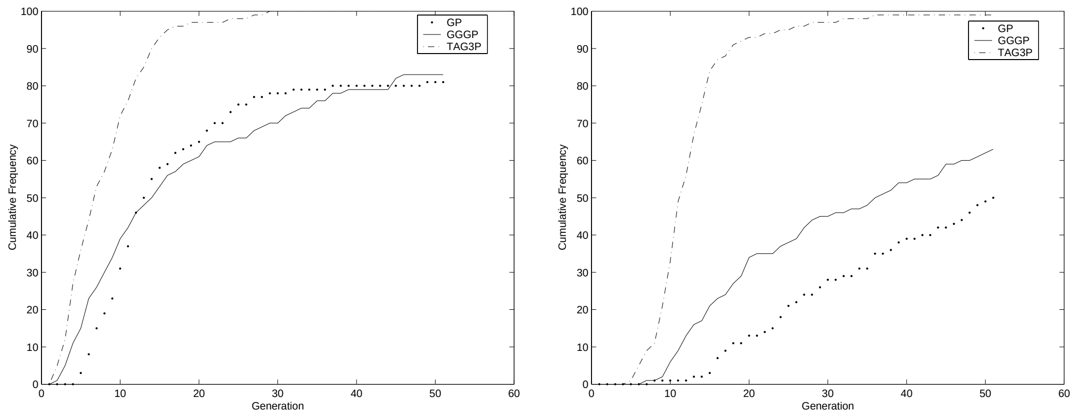


Figure 5.5: Cumulative Frequencies for 6-Multiplexer Problem



$Cos\ x + 2x + 1$ $4x^3 + 3x^2 + 2x + 1$

Figure 5.6: Cumulative Frequencies for Symbolic Integration Problem

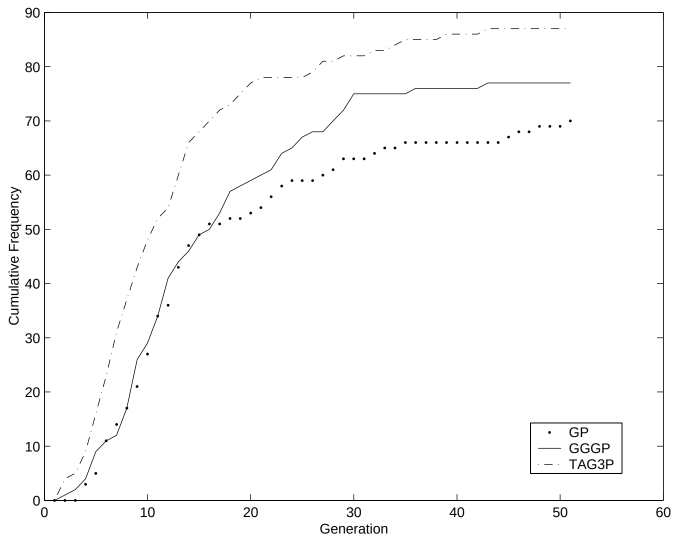


Figure 5.7: Cumulative Frequencies for Symbolic Differentiation Problem

complexity bounds of their individual programs. This problem is investigated in more detail in chapter 10.

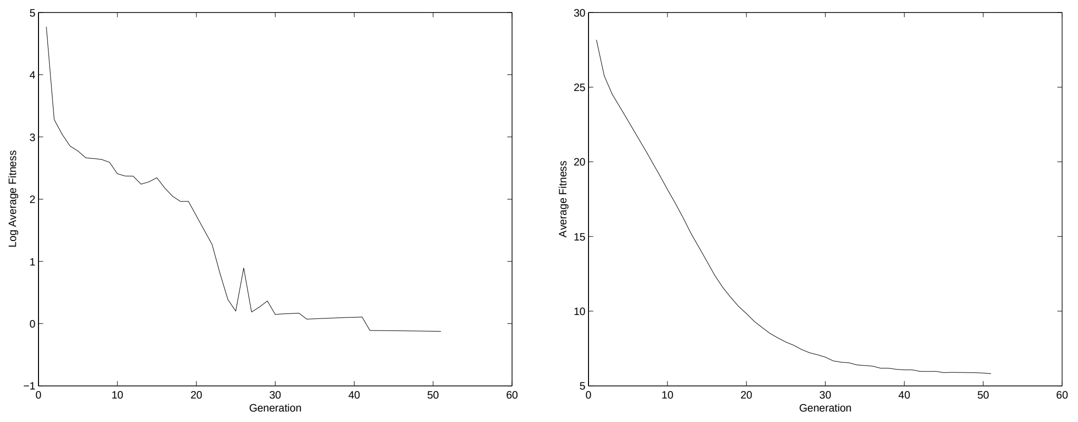
In the following section, some further analyses of the TAG3P runs on two of the eight problem instances above are given and discussed.

5.4 Some further analyses on TAG3P

In this section, some further analyses of the runs in the previous section are presented. Some properties of TAG3P inherited from the TAG-based representation mentioned in the previous chapter are also used to give more information on the behaviour of TAG3P during the evolutionary process. The runs of two problem instances used are 6-multiplexer and symbolic regression with function F_2 .

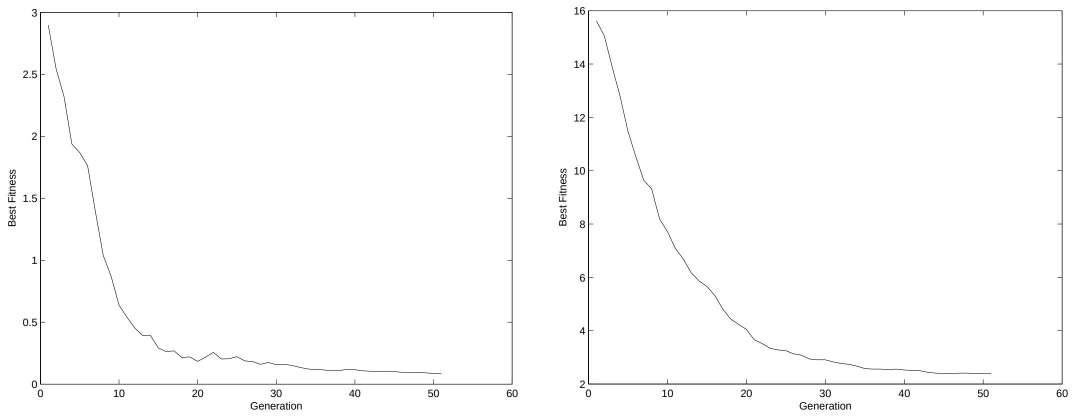
Figures 5.8 and 5.9 depict the time series over generation of the average fitness of the population and the average fitness of the best individual for the two problems. These figures show that the TAG3P population converges to better and better fitness over time.

As mentioned in the previous chapter, thanks to the non-fixed arity property



Symbolic regression 6-Multiplexer

Figure 5.8: Average Fitness of the population



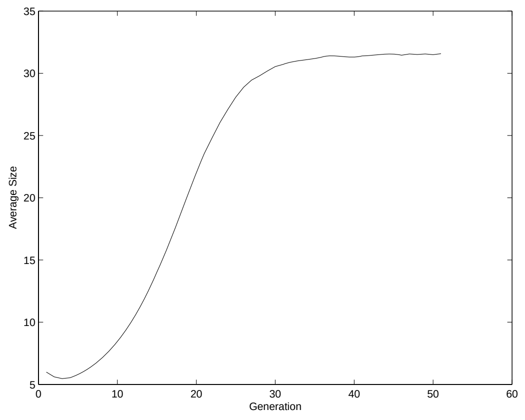
Symbolic Regression 6-Multiplexer

Figure 5.9: Average fitness of the best in the population

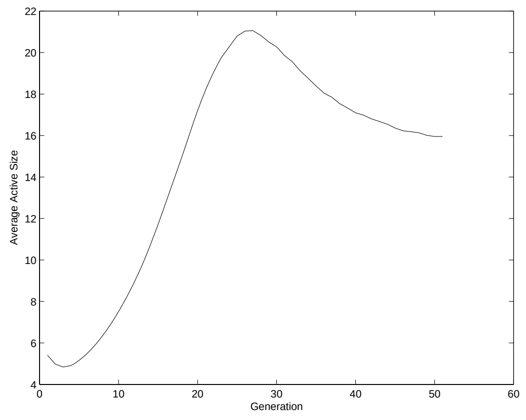
in the TAG-based representation, it is also possible to measure how each subcode (subtree) contributes to the overall fitness of the program containing it. Therefore, it is possible to conduct a study on the subcode level in order to gain more understanding of the dynamics of TAG3P during the evolutionary process.

The first study is to investigate the complexity ratio between active subcode and the whole individual. A subcode is called "inactive" if, when it is removed, the overall fitness of the individual containing it does not change; otherwise, it is called an active subcode. Figures 5.10 and 5.11 show, on the two problems, the evolution of program size and active subcode size in the population, averaging over both the whole population and also over just the best individuals.

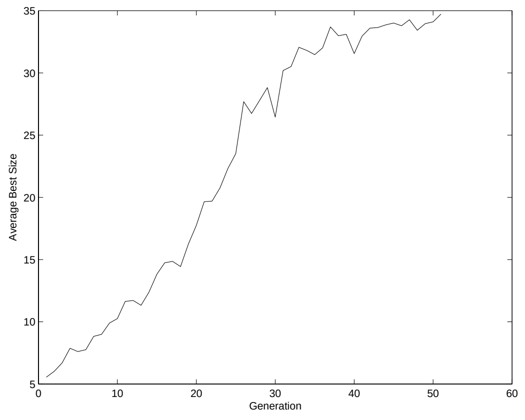
From the Figures 5.10 and 5.11, there is a clear trend for programs in TAG3P to bloat to the maximal allowed size. The size of active (fitness contributing) subcodes evolves on a different trend. For the first half of the evolutionary process, the size of active subcodes increases (both for the best individual and for the population). Presumably, this is because most of the solutions are found in the first half of the evolutionary process, the active size of the subcode needs to increase to accumulate useful subcodes, which are partial solutions. Indeed, by examining some typical successful runs, we found that the most frequent partial subcodes appearing in the best subcodes detected by the methods described in the previous chapter, during this period of evolution, were $X+(X*)$ or $X+(*X)$ for the symbolic regression problem, and $if(a0, d1, *)$ for the 6-multiplexer problem. These are all components of full solutions. For the second period, after the solutions were found, there is a pressure toward solutions with smaller and smaller sizes of active subcodes. This can be explained by the schema theorem in chapter 7, where the active subcode belongs to some useful schemata, and in order to survive they need to be guarded by more and more inactive subcode, so that the ratio $o(H)/n(H)$ ($o(H)$ is the size of the schema and $n(H)$ is the size of the individual that matches H) increases. This is some what similar to conclusion on bloat in [Ban1998]. It is noted that the size ratio of active subcodes



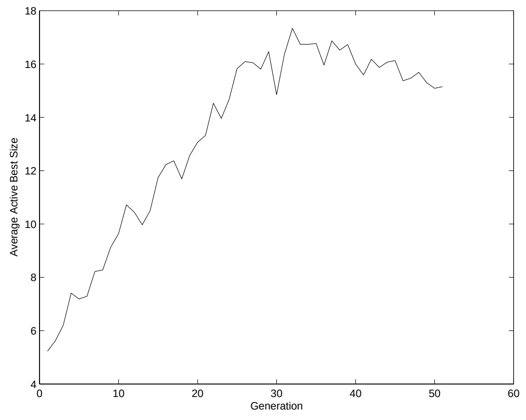
Average size for
the whole population



Average sizes of active
subcodes in the whole population

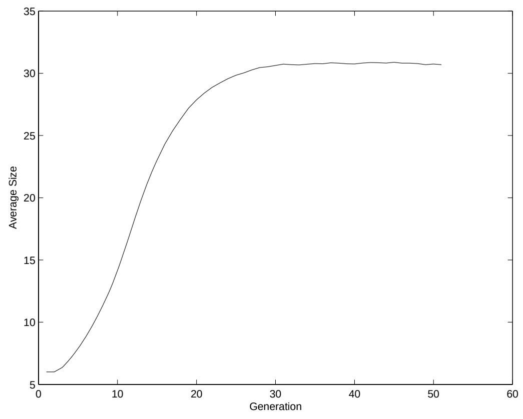


Average size of
the best individual

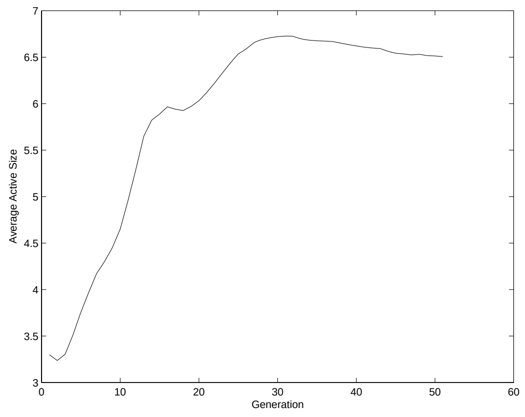


Average sizes of active
subcodes in the best individual

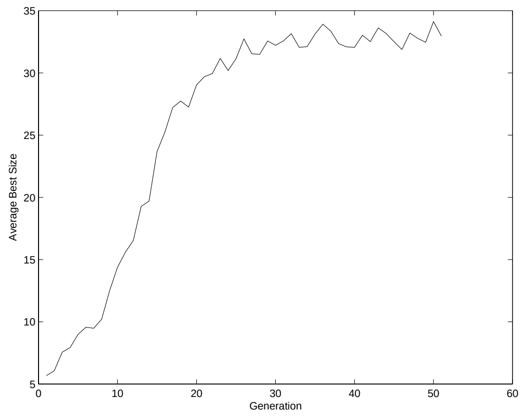
Figure 5.10: Evolution of size in the runs for the symbolic regression problem



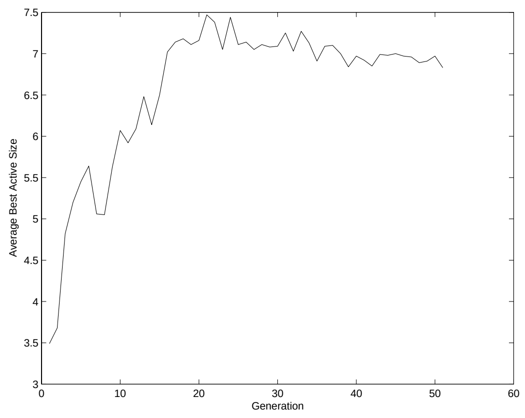
Average size for
the whole population



Average sizes of active
subcodes in the whole population



Average size of
the best individual



Average sizes of active
subcodes in the best individual

Figure 5.11: Evolution of size in the runs for the 6-Multiplexer problem

versus the whole individual is much smaller in the 6-multiplexer problem than in symbolic regression. The reason for this is that Boolean domains have more redundant structures than numerical domains. The average complexity ratio for the solution when first found is 0.947% in the symbolic regression problem, and 0.423% in the 6-multiplexer problem.

5.5 Conclusion

In this chapter, the robustness of TAG3P was tested against a number of standard problem instances, where results were compared with standard GP and CFG-GP. It has been shown that for the problem instances in this chapter, TAG3P is very competitive. It outperformed GP and CFG-GP for almost all the problem instances tried.

The ability to usefully work with subcode fitness described in the previous chapter also helped to extract useful information from the runs in order to gain insight into the dynamics of the TAG3P evolutionary process.

Chapter 6

Some Operators

In any intelligent search system, the search operators play an important part. Usually, search operators are very much representation dependent. Moreover, in the field of evolutionary computation, there is a desire to design and implement genetic operators that simulate the operations of genes in genome evolution [Bac2000a, Bac2000b]. In chapters 3 and 4, it was argued that one of the advantages of TAG-based representation over other GP and GGGP tree-based representations is the non-fixed arity property. Thanks to that property, a number of operators, which are difficult to implement in GP and GGGP, can now be implemented. Some of these operators are bio-inspired.

In this chapter, we show the usefulness of some of the bio-inspired operators, which have been mentioned briefly in the context of the TAG3P system designed in chapter 4. The investigation and analyses will look at two possible uses of those operators: as mutation and generic-local search operators. Some of these research results have been published in [NXH2004b, NXH2004d, NXH2005].

In the following sections, these operators are investigated in three groups. The first group consist of insertion and deletion, the second is relocation, and the last is duplication and truncation.

6.1 Insertion and Deletion Operators

In genome evolution ([Rid1996]), gene insertion and deletion is the addition or removal of a trunk of genetic codes to or from the genotype. The use of gene insertion and deletion is motivated by the use of variable-length genotypes in natural and simulated evolution. From the early days of the field of evolutionary algorithms, which were very much inspired by natural evolution, researchers have employed some forms of gene insertion and deletion operators. For instance, Fogel et al ([Fog1966]) implemented a kind of genetic insertion and deletion by using random operations likes "add a state" and "delete a state" on variable-length genotypes, when they involved finite-state machines.

In the field of genetic programming, when the length (or effective length) of genotypes is variable, gene insertion and deletion are potentially important. Compared to its predecessor, GAs, whose length and structure of genotypes are usually fixed, GP genotype space is much more complicated, since it has three dimensions, namely, content, length (size), and structure. Therefore, it is desirable that insertion and deletion operators can help genetic programming to explore the genotype space in all dimensions. In [Koz99, Koz1995a, Koz1995b], gene deletion was used in a multi-part program representation, where deletion is the removal of a randomly chosen automatically defined functions (ADF) in the function branch. However, gene deletion implemented in that way can lead to the effect that the amount of changes on the program structure is not bounded and controllable. The implementation of genetic deletion directly on expression tree representation in standard GP is more difficult. Recently, Vanneschi et al ([Van2003a, Van2003b]) have defined two new structure-altering operators on GP-expression tree representation, which they name inflate and deflate mutations. However, since their definition is based on the concept of incrementing/decrementing arity (number of children) of primitive sets in GP-expression trees, the operators will become meaningless if all the functions have the same arity. Moreover, with the arity-dependent definition, it is difficult to extend their ideas to grammar guided genetic programming, because derivation trees in grammars are rule-constraint trees rather than trees of different arity nodes.

In grammar guided genetic programming, we believe that it is even harder than GP to implement gene insertion and deletion, since derivation trees of grammars are more constrained with rules from the formalism. Any attempt to add or remove part of an derivation tree might easily result in an invalid derivation tree, with the possible exception of some very ad-hoc and grammar-dependent approaches. In GE, because of the linearity of the genotype, it is possible to implement gene insertion and deletion [Rya1998a]. However, as discussed in chapter 2, the degree of change on the phenotype can be unbounded and not controllable, because the GE representation does not have the locality property. To date, there has not been any work investigating the use of gene insertion and deletion in GE. In the next subsection, two operators, insertion and deletion, on TAG-based representation are defined, and their usefulness in the context of TAG3P is shown.

6.1.1 Description of Insertion and Deletion Operators

Thanks to the non-fixed arity property, gene insertion and deletion can be implemented easily in TAG-based representation. For each node in a TAG derivation tree, its maximal arity is the maximal number of children it can have. In other words, the maximal arity of that node is the number of adjoining addresses in the elementary tree represented by that node. Let's suppose that it is n. Then, because of the non-fixed arity property, in a TAG derivation that node might have any of $0, 1, ..., n$ children. In the case that the number of children of that node in a TAG derivation tree is m and $m < n$, it is said that there are $m - n$ NULL-adjunctions in that node, i.e. there are $m - n$ adjoining addresses in the elementary tree represented by that node, that are not yet adjoined by any other elementary trees. More specifically, the insertion and deletion operators can be implemented as follows.

Insertion Operators. For an individual (a TAG G_{lex} derivation tree) t:

```
1)  Find the set $V$ defined as
    $V$={nodes $n$ in $t$/ $n$ has at least one NULL-adjunction}
2)  Choose $m$ uniformly randomly from $V$.
3)  Choose a NULL-adjunction in $m$ at random.
4)  Among all $\beta$-tree that can adjoin at this NULL-adjunction
    (adjoining address),choose a tree $\beta_1$ at random.
5)  Adjoin $\beta_1$ to the selected NULL-adjunction.
```

Figure 6.1 depicts how insertion works (the rectangles represent NULL-adjunction).

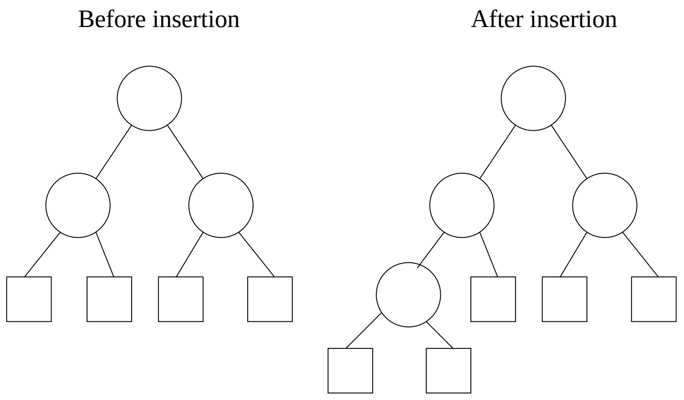


Figure 6.1: Insertion Operator

Deletion Operators. The deletion operator works in exactly the opposite way to the insertion operator. For an individual (a TAG G_{lex} derivation tree) t.

```
1)  Find the set $V$ defined as
    $V$={nodes $n$ in $t$/$n$ has all NULL-adjunction}
2)  Choose $m$ randomly from $V$.
3)  Delete $m$ from $t$.
```

geneThe deletion operation is shown in Figure 6.2.

Essentially, the insertion and deletion operations are the addition and deletion of one leaf node to or from the TAG-derivation tree representing an individual program. They are just the simulation of growth and shrinkage of natural trees. Moreover, the degree of change on genotype level (G_{lex} derivation trees) is minimal (i.e. the distance between tree t and the resultant tree t' being the resultant

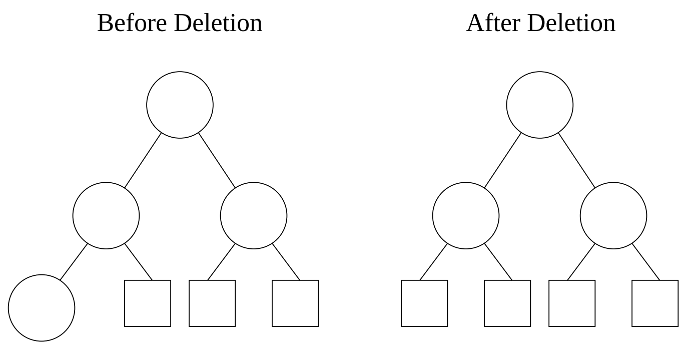


Figure 6.2: Deletion Operator

tree after insertion or deletion is 1 (using the tree distance defined in chapter 3). Also, because the genotype-phenotype map in TAG-based representation has the locality (causality) property, the degree of change on phenotype level is also bounded (i.e. the distance between the derived tree of t and the derived tree of resultant tree t', being the resultant tree after insertion or deletion, is bounded by the maximal number of nodes in an elementary tree of G_{lex}). This property, which is formally proven in chapter 9, allows control of the amount of change, both on genotype and phenotype level, by repeatedly applying insertion and/or deletion a number of times. It is also noted that, when using insertion and deletion as dual twin operators (i.e. they are used as a combined operator with 50% chance each will be applied), the size (length) of the individual will remain fairly constant.

The next subsection shows the usefulness of insertion and deletion when used as a dual twin operator in the context of TAG3P. Their capability in solving the problem of structural difficulty in genetic programming is shown in chapter 8.

6.1.2 Experiments

To investigate the usefulness of insertion and deletion in solving problems, a number of problems were chosen for TAG3P to solve while using those two operators. Since the nature of insertion and deletion is to make minimal changes on genotype space and bounded changes on phenotype space, they might be useful for problems for which a small change to an individual can affect and lead to a small change to its fitness. Therefore, two families of problems, namely, ORDER and MAJORITY from [Gol1998, ORe1998] are chosen. In addition, two standard GP problems from the previous chapter, simple symbolic regression and 6-multiplexer, are also used.

Test Problems

In [Gol1998, ORe1998], two families of test-suite problems for genetic programming, ORDER and MAJORITY, were designed. They are claimed to be GP-versions of one of the popular test-suite problems in GA literature, the ONE-MAX problem ([Mit1996, Ree2003]).

In the ORDER problems, the function set for GP consists of only one function {Join} and the terminal set consists of $2 \times n$ (the size of the problem) possible leaves, $P_1, P_2, ..., P_n$ and $N_1, N_2, ...N_n$. For a GP expression tree, a leaf is called expressed if it is labeled as P_i, for some i from $1, 2, .., n$, and there is no leaf node labeled with either P_i or N_i, which appears before P_i in the pre-order traversal of that tree. The fitness of an individual (GP expression tree) is the number of expressed leaves in that individual. The task is to find a tree with the maximal fitness, which is n.

The MAJORITY problems have the same function and terminal sets as ORDER. However, a leaf, in an individual (GP expression tree) is called expressed if and only if it is labeled with P_i, for some i from $1, 2, .., n$; and if it is the first leaf (in pre-order traversal of the tree) that is labeled with P_i; and, the number of occurrences of leaves labeled with P_i is bigger than the number of occurrences of leaves labeled with N_i in the individual. The task is to find a tree with maximal fitness, which in that case, as in the ORDER problems, is n.

The ORDER and MAJORITY problems have some interesting aspects. Firstly, they are two families of problems, where the difficulties are scalable. Secondly, small changes on one tree, such as “add a leaf” and/or “delete a leaf”, might lead to (small) changes of its fitness. Finally, although they are artificial problems, they share a number of properties with other standard and real world problems for genetic programming [Gol1998].

The simple symbolic regression (SYMREG) and 6-multiplexer (6MUL) problems were described in the previous chapter. For the simple symbolic regression problem in this chapter, the function F_2 (the quadtic polynomial) is used as the target function.

Experiment Setup

To investigate the usefulness of the insertion and deletion operators, two possible ways of using them on test problems were studied, namely, as the mutation operator for TAG3P, and as generic-local search operators in combination with the genetic search in TAG3P. The reason for them being called generic-local search operators, is to distinguish them from the problem-specific local search operators used in combinatorial optimisation [Aar1997, Hro2003]. The design of local search operators used in combinatorial optimisation is usually problem dependent, and also search space representation dependent ([Aar1997]). Examples of such kinds of local search operators include the well-known 2-opt and 3-opt local search operators for the traveling salesman problem based on Lin and Kernighan's heuristic [Lin1973]. On the contrary, insertion and deletion operators, as defined on TAG-based representation in this chapter, are only dependent on the representation. They are problem independent.

The performance of TAG3P using insertion and deletion operators was compared against a control of TAG3P using standard subtree crossover and subtree mutation. The settings of the TAG3P base runs for simple symbolic regression and 6-multiplexer, were exactly the same as in the previous chapter (such as POPSIZE=500, MAXGEN=51, MAXSIZE=40,...). For the ORDER and MAJORITY problems, the setting are summarised in Table 6.1. As a reference, the same number of base runs was also conducted for GP with similar settings.

Table 6.1. Tableau for MAJORITY and ORDER problems.

Objective	Find a tree with maximal possible fitness.
Terminal set	$P_1, P_2, .., P_n; N_1, N_2, .., N_n$
Function set	Join.
Fitness cases	None.
Raw fitness	Number of expressed leaves
Standardised fitness	n-Raw fitness.
Hits	Equivelent to raw fitness.

Parameters	POPSIZE=100, MAXGEN=51,
	Tournment selection with size 3.
	MAXSIZE=1000 for TAG3P, MAXDEPTH=15 for GP.
	Crossover rate=0.9, mutation rate=0.1.

Success Predicate	A program scores n hits.

The grammars (G and G_{lex}) for the simple symbolic regression problem and 6-multiplexer problems are the same as in the previous chapter, while the grammars for the ORDER and MAJORITY problems are given in Appendix A.

Experiment 1

In the first experiment, the insertion and deletion operators were used as mutation operators in TAG3P (TAG3PM) and the results were compared with standard GP and TAG3P using subtree mutation. For the ORDER and MAJORITY, four problem sizes were used, $n = 25, 30, 35, 40$ (ORDER25, ORDER30, ORDER35, ORDER40, MAJ25, MAJ30, MAJ35, MAJ40). To separate out the effect of using insertion and deletion as mutation operators from the pure power of subtree crossover in TAG3P, a set of runs was dedicated to TAG3P (TAGCROSS) using subtree crossover as the sole genetic/search operator. For each of these system and problem instances, 100 runs was allocated, making a total number of 4000 runs.

Table 6.2 below shows the proportion of success on the ten problem instances.

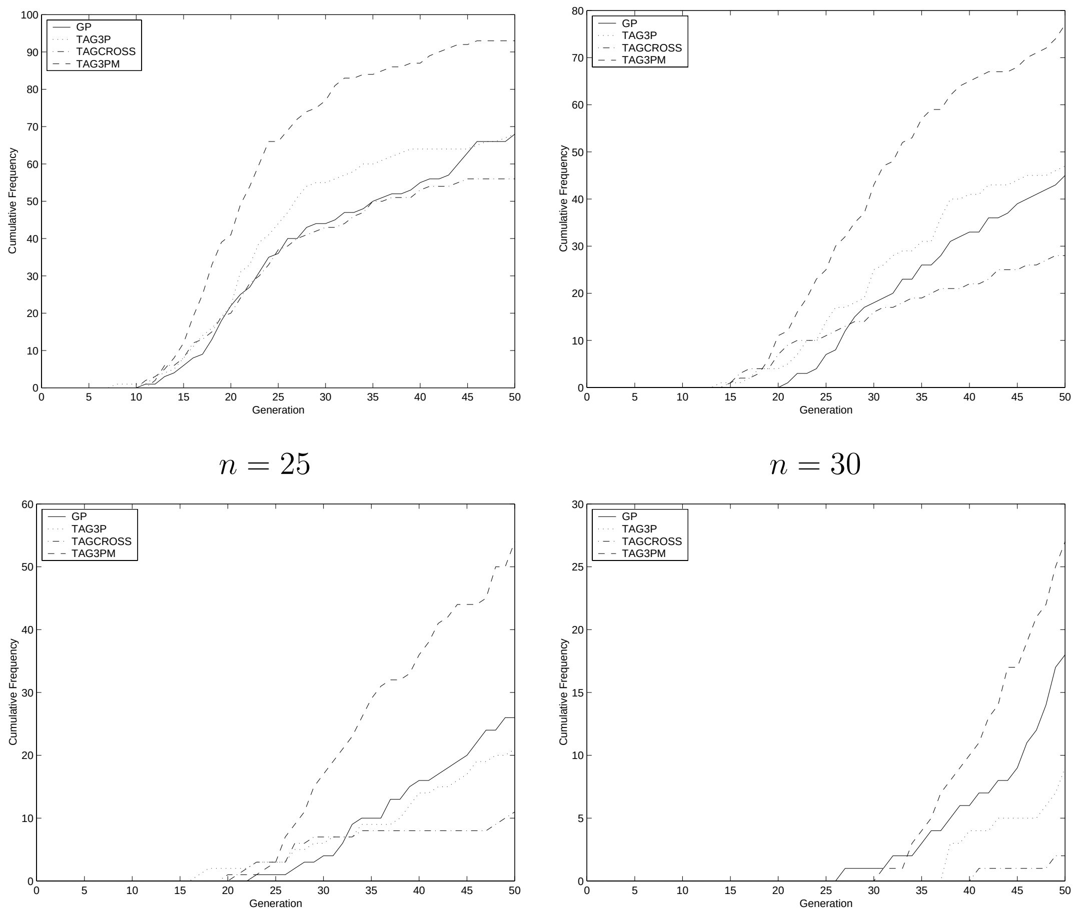
Figures 6.3, 6.4, 6.5, and 6.6 depict the cumulative frequencies of GP, TAG3P, TAGCROSS, and TAG3PM on all problem instances.

Table 6.2. Proportion of success on test problem instances.

Problem	GP	TAG3P	TAGCROSS	TAG3PM
ORDER25	68%	68%	56%	93%
ORDER30	45%	47%	28%	77%
ORDER35	26%	21%	11%	54%
ORDER40	18%	9%	2%	27%
MAJ25	73%	75%	55%	88%
MAJ30	46%	52%	30%	71%
MAJ35	25%	25%	8%	53%
MAJ40	15%	14%	5%	25%
SYMREG	9%	93%	93%	90%
6MUL	63%	63%	61%	79%

The results in table 6.2 and Figures 6.3, 6.4, 6.5, and 6.6 show that for almost all of the problem instances tried, insertion and deletion are a much better mutation operator than subtree mutation in TAG3P. They helped TAG3PM to converge to solutions more quickly and outperformed TAG3P and GP. The superior performance of TAG3PM over TAG3P and GP is statistically significant (using one-tailed statistical test on the difference between two random binomial variables with $\alpha = 0.05$) on 7 out of 10 problem instances. Moreover, compared to TAGCROSS on these problem instances, it is clear that the superior performance of TAG3PM over TAG3P and GP comes from the use of insertion and deletion as mutation operators but not from the power of subtree crossover.

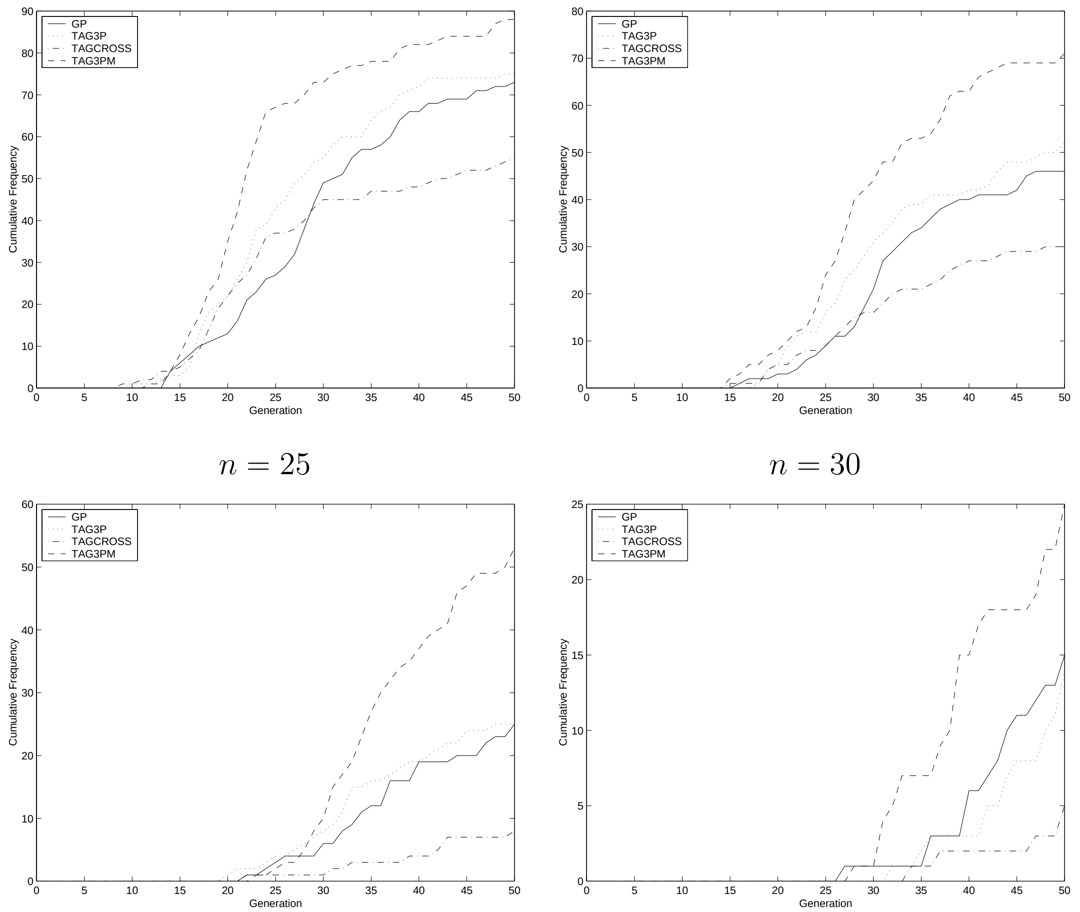
The three exceptions are ORDER40, MAJ40, SYMREG. For the symbolic regression problem (SYM), the results of TAGCROSS shows that the success rate of TAG3P was very high due to subtree crossover. Therefore, it is not surprising that TAG3PM had a similar result with TAG3P and TAG3PCROSS. For ORDER40 and MAJ40, TAG3PM outperformed TAG3P and GP slightly, but not statistically significantly. Those results, and the decline in performance



$n = 25$ $n = 30$

$n = 35$ $n = 40$

Figure 6.3: Cumulative Frequencies for ORDER Problems



$n = 25$ $n = 30$

$n = 35$ $n = 40$

Figure 6.4: Cumulative Frequencies for MAJORITY Problems

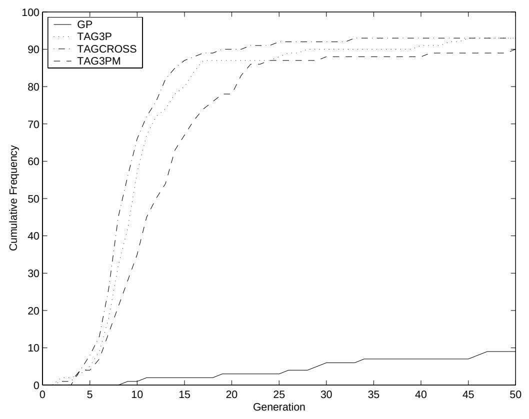


Figure 6.5: Cumulative Frequencies for symbolic regression Problem

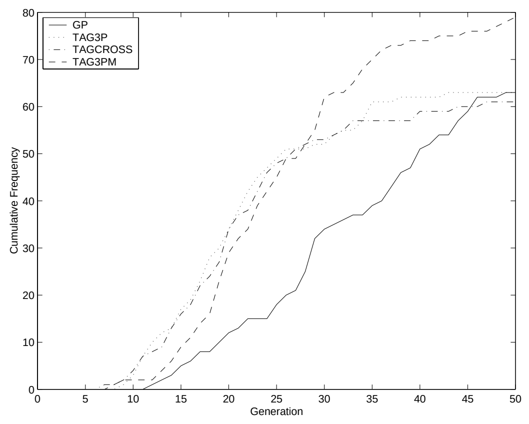


Figure 6.6: Cumulative Frequencies for 6-Multiplexer Problem

of TAG3PM on the other ORDER and MAJORITY problem instances, might indicate that although insertion and deletion are better mutation operators for TAG3P on those problems (MAJORITY and ORDER), they do not scale well against increasing size (i.e. increasing difficulty) of the problems.

Experiment 2

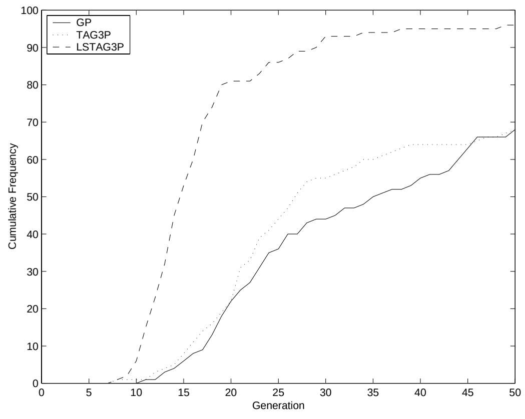
In the second experiment, insertion and deletion are used as a dual meta local search operator in combination with TAG3P genetic search (LSTAG3P) using subtree crossover and subtree mutation. The results are compared with TAG3P (TAG3P) using subtree crossover and subtree mutation. To ensure that the number of fitness evaluations was the same between LSTAG3P and TAG3P the population size in LSTAG3P was reduced to a very small size. For the ORDER and MAJORITY problems, the population size of LSTAG3P was set as 10 (LSTAG3P10). Correspondingly, the number of local search steps was 10. For symbolic regression and 6-multiplexer problems, two sizes of population for LSTAG3P were used, namely, 50 (LSTAG3P50) and 10 (LSTAG3P10). Therefore, the number of local search steps for these were 10 and 50 respectively. Other settings of LSTAG3P (such as MAXGEN, MAXSIZE, operator probabilities,...) were exactly the same as TAG3P. The local search strategy used was stochastic hill-climbing. Lamarckian inheritance was used (i.e. whenever a better individual was found in the neighbourhood of an individual in the population, this individual was replaced by the new and better individual). For each setting of each system, some 100 runs were conducted, thus making the total number of runs for this experiment 1200.

The following table shows the proportion of success for all systems. For the sake of reference, GP results are also given. Figures 6.7, 6.8, 6.9, and 6.10 depicts their cumulative frequencies.

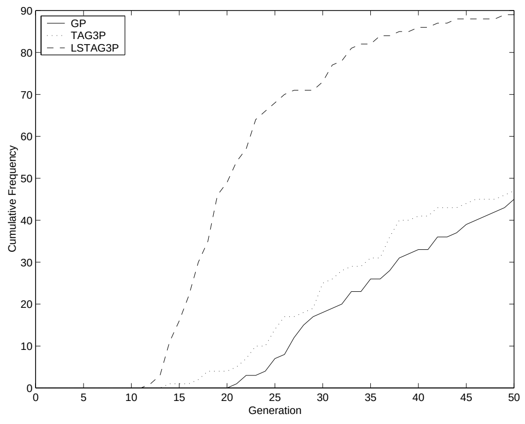
Table 6.3. Proportion of success on test problem instances.

Problem	GP	TAG3P	LSTAG3P50	LSTAG3P10
ORDER25	68%	68%	N/A	96%
ORDER30	45%	47%	N/A	89%
ORDER35	26%	21%	N/A	89%
ORDER40	18%	9%	N/A	80%
MAJ25	73%	75%	N/A	93%
MAJ30	46%	52%	N/A	92%
MAJ35	25%	25%	N/A	92%
MAJ40	15%	14%	N/A	79%
SYMREG	9%	93%	78%	55%
6MUL	63%	63%	59%	64%

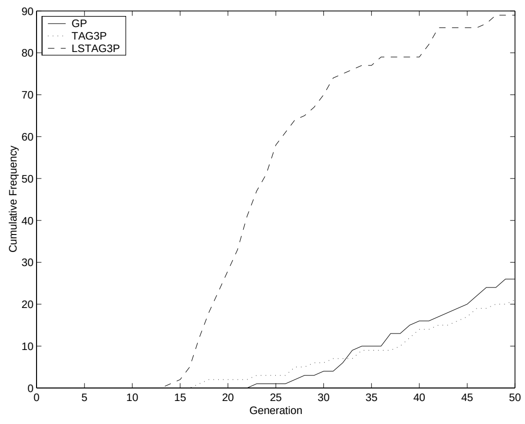
The results shown in table 6.3 and Figures 6.7, 6.8, and 6.9, 6.10 demonstrate that LSTAG3P10 outperformed GP and TAG3P statistically significantly on the ORDER and MAJORITY problems (using the one-tailed statistical test for the difference between two binomial random variables with $\alpha = 0.01$). Moreover, LSTAG3P performance scaled very well (compared to GP, TAG3P, and TAG3PM) against the increasing problem complexity. The performances of both LSTAG3P50 and LSTAG3P10 are similar to or worse than TAG3P (and GP) on the 6-multiplexer and symbolic regression problems respectively (though LSTAG3P50's performance was not very far behind TAG3P). However, it is noted that the population sizes used in the LSTAG3P runs are very small compared to the normal population sizes in the genetic programming literature. The



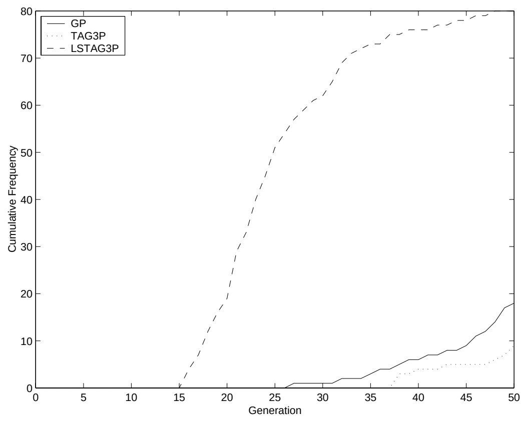
$n = 25$



$n = 30$

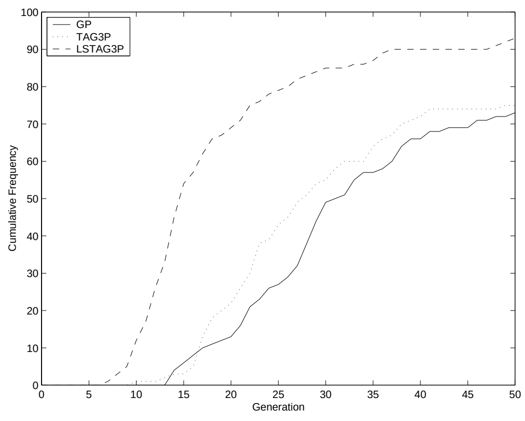


$n = 35$

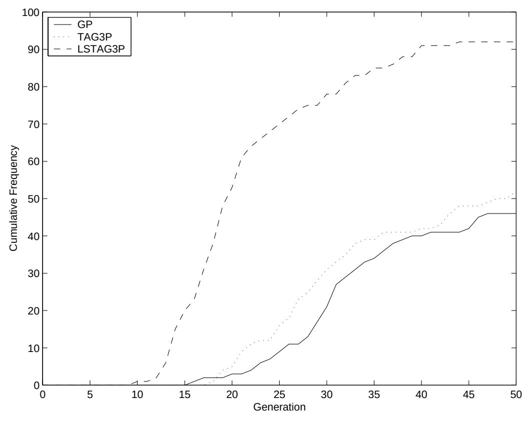


$n = 40$

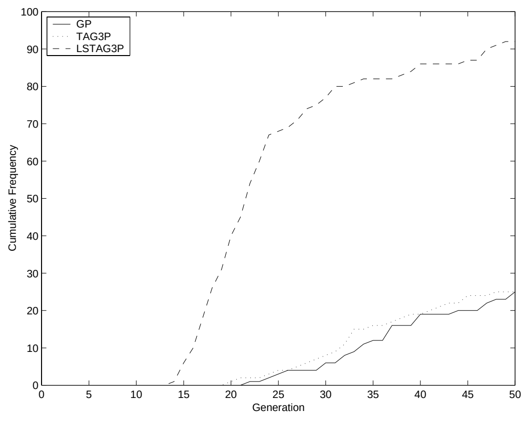
Figure 6.7: Cumulative Frequencies for ORDER Problems



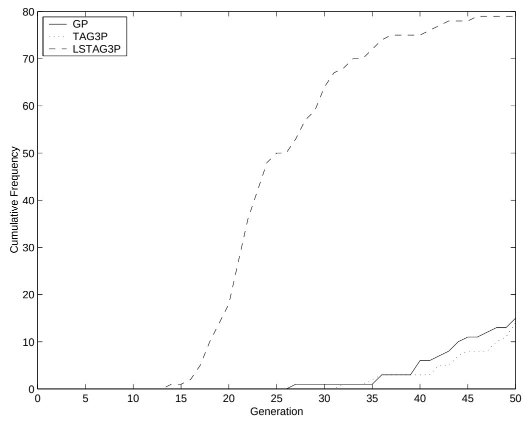
$n = 25$



$n = 30$



$n = 35$



$n = 40$

Figure 6.8: Cumulative Frequencies for MAJORITY Problems

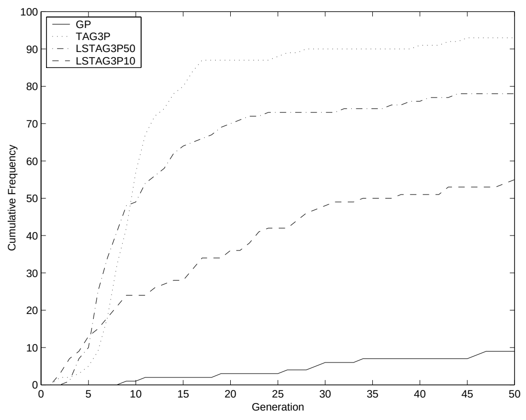


Figure 6.9: Cumulative Frequencies for symbolic regression Problem

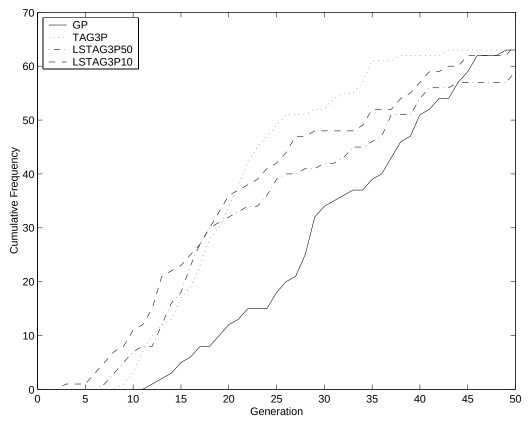


Figure 6.10: Cumulative Frequencies for 6-Multiplexer Problem

power of LSTAG3P and the similarity of LSTAG3P50 and LSTAG3P10 on the 6-Multiplexer problem indicate that using insertion and deletion as a dual generic-local search operator can help LSTAG3P to achieve a better or at least similar performance with TAG3P (and GP) even with very small population sizes. It might be hoped that insertion and deletion are useful, in real world applications, where very big population sizes of GP are usually required. For instance, in [Koz99], population sizes of hundreds of thousand to tens of millions were used. If the problems possess some heuristics indicating that insertion/deletion might work well (such as "small change in genotype will likely cause small change in phenotype and its fitness" as in ORDER and MAJORITY), then it might be possible to solve them with very small population sizes by using insertion/deletion as a generic-local search operator, and, therefore, reduce the memory storage requirements.

Experiment 3

In any evolutionary algorithm (EAs), there is always a trade-off between exploitation and exploration capability ([Hol1975]). One of the many possible ways for EAs to shift the search process toward exploitation is to use a smaller population size with bigger number of generations (longer evolution time). Therefore, the purpose of experiment 3 is to show that the result from experiment 2 really do come from the power of insertion/deletion as a meta-local search operator and not from the effect of increasing the exploitation of TAG3P by reducing the population sizes. To achieve that goal, TAG3P was run with population sizes of 50 (TAG3P50) and 10 (TAG3P10) with the correspondingly longer number of generations. The maximal number of generations (MAXGEN) of TAG3P10 was 511 for the ORDER and MAJORITY problems; an 2551 for the symbolic regression, and 6-multiplexer problems. Similarly, MAXGEN of TAG3P50 for symbolic regression problem and 6-multiplexer problems was 511. Consequently, TAG3P10 and TAG3P50 had the same maximal number of fitness evaluations as other systems in experiment 1 and 2. Similarly, the same settings were used for GP (GP50, GP10) and the results of those GP runs are also shown as a reference.

Each of the settings was allocated 100 runs, which makes the number of runs for this experiment as 2400. The following table depicts the results of TAG3P10 and TAG3P50 (compared to GP50, GP100 and LSTAG3P10, LSTAG3P50).

Table 6.4. Proportion of success on the test problems.

Problem	GP50	TAG3P50	LSTAG3P50	GP10	TAG3P10	LSTAG3P10
ORDER25	N/A	N/A	N/A	55%	29%	96%
ORDER30	N/A	N/A	N/A	37%	19%	89%
ORDER35	N/A	N/A	N/A	13%	12%	89%
ORDER40	N/A	N/A	N/A	12%	7%	80%
MAJ25	N/A	N/A	N/A	49%	27%	93%
MAJ30	N/A	N/A	N/A	30%	21%	92%
MAJ35	N/A	N/A	N/A	12%	16%	92%
MAJ40	N/A	N/A	N/A	10%	3%	79%
SYMREG	12%	70%	78%	4%	51%	55%
6MUL	52%	19%	59%	28%	15%	64%

The results show that LSTAG3P50 (for 6-Multiplexer problems) and LSTAG3P10 significantly outperformed TAG3P50 (GP50) and TAG3P10 (GP10) respectively. The only exception is on symbolic regression, whereby LSTAG3P50 (LSTAG3P10) was better than TAG3P50 (TAG3P10), but not statistically significantly. Therefore, the power of LSTAG3P clearly comes from the use of insertion/deletion as a dual generic-local search operator, not from the exploitation effect of the reduction in population sizes.

Experiment 4

In GP literature, there has been some works suggesting that local search could sometime defeat genetic search in GP [ORe1994, Lan1995]. Therefore, the purpose of this experiment is to investigate whether the high performance of LSTAG3P was attributed to the use of insertion/deletion as (generic) local search operators or attributed to the local search strategy alone. To accomplish that task, GP (LSGP10) and 50 (LSGP50 - for 6-multiplexer and symbolic regression only) and was combined with local search using subtree mutation as the local search operator. The other parameter settings were the same as in the previous experiments. Each setting was allocated 100 runs, which made the total number of runs as 1200. Table 6.5 depicts the results.

Table 6.5. Proportion of success on the test problem.

Problem	LSGP50	LSTAG3P50	LSGP10	LSTAG3P10
ORDER25	N/A	N/A	82%	96%
ORDER30	N/A	N/A	76%	89%
ORDER35	N/A	N/A	67%	89%
ORDER40	N/A	N/A	60%	80%
MAJ25	N/A	N/A	90%	93%
MAJ30	N/A	N/A	78%	92%
MAJ35	N/A	N/A	62%	92%
MAJ40	N/A	N/A	52%	79%
SYMREG	6%	78%	7%	55%
6MUL	64%	59%	41%	64%

The results from Table 6.5 show that except for MAJ25 and 6-MUL, LSTAG3P outperformed LSGP significantly on all problem instances. For MAJ25, LSTAG3P10 was just slightly better than LSGP10 (not statistical significance). On the other hand, LSGP50 performed better than LSTAG3P with a very small margin. For population size of 10, LSTAG3P10 outperformed LSGP10 statistically significantly. Consequently, the results show that the good performance of LSTAG3P did really come from the usefulness of insertion/deletion operators when being used as local operators but not from the local search strategy itself.

6.1.3 Insertion and Deletion Conclusion

On the problems tried, insertion and deletion work well both as mutation operators and as a dual generic-local search operator combining with genetic search. However, when acting as mutation operators insertion and deletion did not perform well when the problem complexity is scaled up. On the contrary, when using insertion and deletion as a dual genetic-local search operators on relevant problems, it not only helped TAG3P to perform well in general, but also scaled well against the problem complexity, using very small population sizes.

6.2 Relocation Operator

Genetic transposition in genome evolution is a phenomenon whereby a region of DNA copies itself to another place on the genome. These mobile genetic sequences are called transposable elements [Drl1984, Rid1996], transposons, or more informally jumping genes [Wat1995]. It is conjectured that genetic transposition plays an important role in forming scattered clusters of related genes in the genome of organisms.

There are two types of genetic transposition, namely, replicative transposition and conservative transposition [Rid1996]. In the former, a transposon makes a repeated copy of itself elsewhere using reverse transcription on an RNA, while in the latter a transposon moves to another place by copying itself [Rid1996]. In this section, we investigate the metaphor of conservative genetic transposition in TAG3P, which we call the relocation operator. Replicative tranposition is studied in the next section.

In genetic programming, genetic relocation can be seen as the process of internal rearrangement of sub-codes (sub-trees in tree-based representation, subsequences in linear representation) in each individual. This rearrangement is very useful in at least two ways. Firstly, if one sees the process of genetic evolution in GP as discovering and accumulating necessary building blocks for the solution(s), then rearrangement within one individual helps to accelerate this accumulation, if some useful genetic materials (sub-codes) already appear in the individual itself. Secondly, the rearrangement of subcode within an individual helps to bring the related and useful subcodes together, thereby protecting them from the destructive effect of the crossover operator [Nor1995a, Nor1995b, Nor1996, Ban1998].

In standard GP, using expression tree representation, it is hard to see how this subcode rearrangement (genetic relocation) can be done. We believe that the expression tree representation makes it difficult to design a general purpose search operator to accomplish this task. As an alternative, standard GP uses mechanisms such as automatically defined functions (ADF) to shorten the solution length, therefore accelerating the accumulation of building blocks for the desired solution during its evolutionary process [Koz94, Koz99]. However, we argue that when the length of the solution is long, the use of genetic relocation provides a more flexible way to accumulate the building blocks needed to solve the problem. Unlike GP with ADF, it does not require users to provide extra information such as the number of ADFs, number of ADF parameters, and so on.

For GEP, its linear representation allows the ready design of genetic relocation (which was called transposition in [Fer2001, Fer2002a]). However, the relocation of any trunk of genes (subcode) in GEP can potentially affect the positions as well as the expressiveness (i.e. coding or non-coding) of many other genes not just at the source and destination positions. The reason for this is, as discussed in chapter 2, that the GEP genotype-phenotype map does not have the locality (causality) property. Therefore, it creates a global random side effect on the phenotype. Nevertheless, it improved the performance of GEP in some cases as shown in [Fer2002a]. Furthermore, in [Fer2002a], it is suggested that it is better to use genetic relocation in combination with crossover.

In grammar guided genetic programming (GGGP), where the language of the programs is dictated by a grammar (usually a string-rewriting grammar), it is even harder to implement genetic relocation on the genotypic level (usually derivation trees of the grammar) because of the rule-based nature of the formalism. GE is an exception. Thank to the linear structure of the genotype, it is easy to implement genetic relocation. However, as with GEP, since the GE genotype-phenotype map does not possess locality (causality) property, the relocation of trunk of genes (subcode) in GE might completely change the meaning (if there is more than one non-terminal symbol in the grammar) and the expressiveness of the genes following the destination position. The meaning of the relocated genes might also change vastly, depending on the context before the destination position. Therefore, as with GEP, it creates a global random side effect on the phenotype. To date, there has not been any concrete results in the use of genetic relocation in GE that we are aware of.

6.2.1 Description of the Relocation Operator

In TAG-based representation, the non-fixed arity property helps to design a relocation operator in a simple and natural way. Moreover, since TAG-based representation has the locality property, the movement of a particular subcode does not affect the meaning of others. Therefore, it is hoped that relocation will help to assemble together the useful codes within an individual. The relocation operator on an individual, a G_{lex} derivation tree t, can be implemented as follows.

```
1)  Choose a subtree $v$ (subcode) in $t$ as random.
    Assume that $v$ rooted with node $l$.
2)  Find the set $V$ defined as
    $V$={nodes $n$ in ($t$-$v$)/$n$ has at least
    one NULL-adjunction that $l$ can adjoin into}
3)  IF $V \neq \emptyset$ THEN
     Choose $m$ uniformly randomly from $V$.
     Randomly choose an address $a$ among all the adjoining addresses
     in $m$ that $l$ can adjoin into.
     Detach $v$ from $t$.
     Attach $v$ to $t$ at $m$ with adjoining address $a$.
     Exit.
4   Loop from 1 to 3 until a maximal number of attempts is exceeded.
```

where (t-v) means all nodes in tree t that are not in subtree v. Figure 6.11 demonstrates how the relocation operator works. Note that the relocation operator does not change the size of the individual.

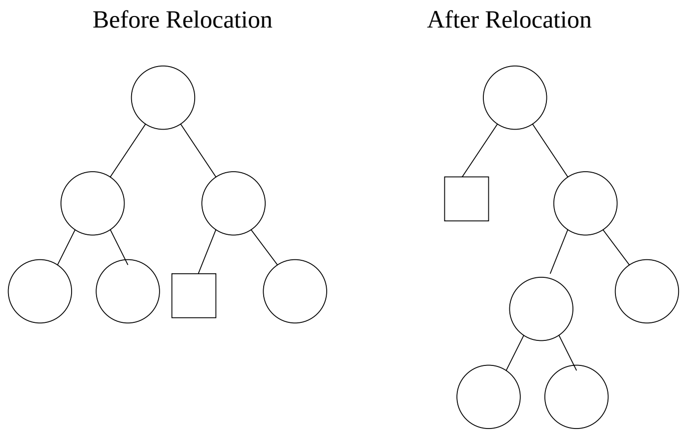


Figure 6.11: Relocation Operator

6.2.2 Experiments

As with the insertion and deletion operators, in this subsection, the role of the relocation operator is investigated, in the context of TAG3P, in two ways. The first role will be as a mutation operator, while the second will be as a generic-local search operator. Therefore, some relevant standard problems from GP literature were chosen as test problems. Then the experiments were conducted in a very similar way as for insertion and deletion operators in the previous section.

Test Problems

The chosen test problems for relocation were learning the quintic and sextic polynomials, trigonometric identities, and two box problems [Koz92, Koz94]. In the first two problems, the tasks was to learn the quintic polynomial (QUINTIC) $X^5 - 2X^3 + X$ and sextic polynomial (SEXTIC) $X^6 - 2X^4 + X$, from sample data. In the third problem (TWOBOX), it is required to learn the formula for calculating the difference in volume between two rectangular boxes from random integer samples of box sizes. The task for the last problem (TRIGO) was to discover the trigonometric identities of $cos\ 2x$ using the sine function.

In [Koz94], the first three problems were used to test the applicability of automatically defined functions (ADFs) in GP. Although the lengths for the solutions in those problems are long, they have some repeated patterns, which can be potentially exploited by using ADFs, therefore making them shorter. However, the results of GP using ADFs was not particularly good on those problems. One possible explanation is that although there are some repeated patterns in the solution, perhaps their number of occurrences are not high enough to be usefully exploited by ADFs [Koz94]. This is the place, we believe, where the relocation operator will play an important role. In the last problem, as pointed out in [NXH2001b], there is a strong tendency for GP search to converge to the approximate solutions, namely, $sin(2x + \pi/2)$ and $sin(\pi/22x)$. However, in order to approximate the constant $\pi/2$, it is necessary to accumulate a long sequence of code (including *,/,+,1.0). Consequently, all four of standard problems above are appropriate as test-beds for investigating the usefulness of the relocation operator in TAG3P.

Experiment Setup

The performance of TAG3P using the relocation operator was compared against TAG3P using standard subtree crossover and subtree mutation, which comprises the set of base runs. Tables 6.6, 6.7, and 6.8 summarise the settings for TAG3P base runs on the four test problems. As a reference, the same number of base runs was also conducted for GP with similar settings. It is noted that the population size (POPSIZE) for the TWOBOX problem was set as 4000, since this problem is considered as a much more difficult problem than other three. In the following section, parameters will, in general, be stated for all problems. However, a shorthand will be used for the population size of the TWOBOX problem: when it is different from other problems it will be enclosed in the curly brackets. The grammars for all problems are given in Appendix A.

Table 6.6. Tableau for QUINTIC and SEXTIC problems.

Objective	Learning quintic (sextic) polynomial from data.
Terminal set	X
Function set	$\{+, -, *, \%\}$
Fitness cases	A random sample of 50 points in the interval [-1..+1].
Raw fitness	The sum, taken over 50 fitness cases, of the errors.
Standardised fitness	Same as raw fitness.
Hits	The number of fitness cases for which the error is less than 0.01.

Parameters	POPSIZE=1000, MAXGEN=51,
	Tournament selection with size 3.
	MAXSIZE=40 for TAG3P, MAXDEPTH=15 for GP.
	Crossover rate=0.9, mutation rate=0.1.

Success Predicate	A program scores 50 hits.

Table 6.7. Tableau for TRIGO problem.

Objective	Discovering identities for $cos2x$ from data.
Terminal set	$\{X, 1\}$
Function set	$\{+, -, *, \%, sin\}$
Fitness cases	A random sample of 20 points in the interval $[0..2\pi]$.
Raw fitness	The sum, taken over 20 fitness cases, of the errors.
Standardised fitness	Same as raw fitness.
Hits	The number of fitness cases for which the error is less than 0.01.

Parameters	POPSIZE=1000, MAXGEN=51,
	Tournament selection with size 3.
	MAXSIZE=40 for TAG3P, MAXDEPTH=15 for GP.
	Crossover rate=0.9, mutation rate=0.1.

Success Predicate	A program scores 20 hits.

Table 6.8. Tableau for TWOBOX problem.

Objective	Learning the function for calculating the difference. between two volumes.
Terminal set	$\{W, H, L, w, h, l\}$
Function set	$\{+, -, *, \%\}$
Fitness cases	A random sample of 10 integers between 1 and 10.
Raw fitness	The sum, taken over 10 fitness cases, of the errors.
Standardised fitness	Same as raw fitness.
Hits	The number of fitness cases for which the error is less than 0.01.

Parameters	POPSIZE=4000, MAXGEN=51,
	Tournment selection with size 3.
	MAXSIZE=40 for TAG3P, MAXDEPTH=15 for GP.
	Crossover rate=0.9, mutation rate=0.1.

Success Predicate	A program scores 10 hits.

Experiment 1

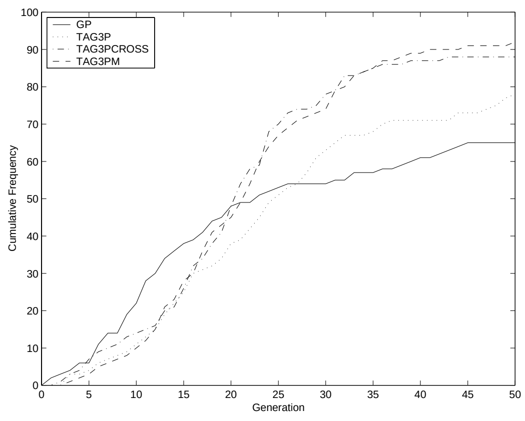
In the first experiment, the relocation operator was used as the mutation operator in TAG3P(TAG3PM) and the results were compared with standard GP and TAG3P (TAG3P) using subtree mutation. To separate out the effect of using relocation as mutation operators, from the pure power of subtree crossover in TAG3P, a set of runs was dedicated to TAG3P (TAGCROSS) using subtree crossover as the sole genetic operator. For each of the system and problem instances, 100 runs were allocated, making the total number of runs as 1600.

Table 6.9 below shows the proportion of success on the four problems. Figure 6.12 shows the cumulative frequencies of GP, TAG3P, TAGCROSS, and TAG3PM on those problems.

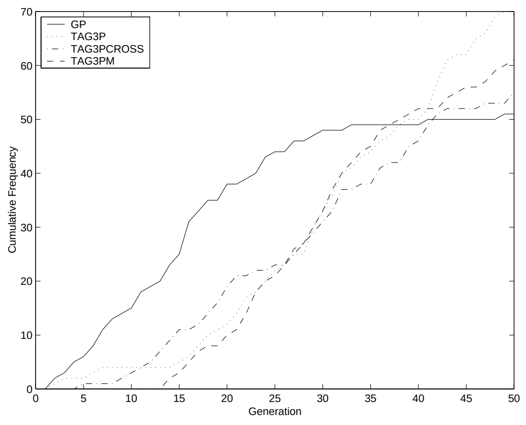
Table 6.9. Proportion of success on the test problems.

Problem	GP	TAG3P	TAGCROSS	TAG3PM
QUINTIC	65%	78%	88%	92%
SEXTIC	51%	70%	55%	61%
TRIGO	36%	36%	22%	47%
TWOBOX	6%	23%	17%	22%

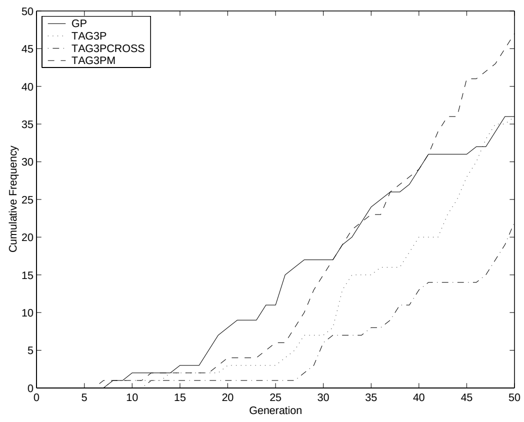
The results indicate that relocation works moderately well as a mutation operator compared to TAG3P using subtree mutation. On SEXTIC and TWOBOX, the TAG3PM performs worse than TAG3P, but the difference is not statistically significant. The use of relocation as a mutation operator made TAG3PM outperform TAG3P (using subtree mutation) statistically significantly on QUINTIC (using one-tailed statistical test of the difference between two random binomial variables with $\alpha = 0.05$) and only slightly (not statistically significantly) on TRIGO. The difference in performance between TAGCROSS and TAG3PM indicates that relocation seems to be a moderately good mutation operator for those problems.



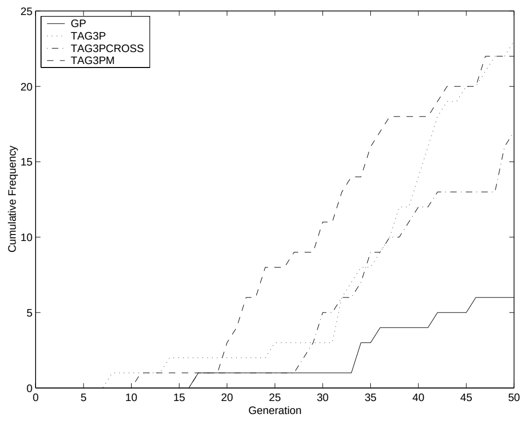
QUINTIC



SEXTIC



TRIGO



TWOBOX

Figure 6.12: Cumulative frequencies for four problems

Experiment 2

In the second experiment, relocation was used as the generic-local search operator in combination with TAG3P genetic search (LSTAG3P) using subtree crossover and subtree mutation. The results were compared with TAG3P using subtree crossover and subtree mutation. To guarantee that the number of fitness evaluation was the same between LSTAG3P and TAG3P, the population size in LSTAG3P was reduced to very small numbers. For the SEXTIC, QUINTIC, and TRIGO problems, the population sizes of LSTAG3P were set as 100 (LSTAG3P100), 50 (LTAG3P50), and 20 (LSTAG3P20). Therefore, the number of local search steps was 10, 20, and 50. For the TWOBOX problem the sizes of population for LSTAG3P were 200 (LSTAG3P200), 100 (LSTAG3P100), and 50 (LSTAG3P50); and the number of local search steps were 20, 40, and 80. Other settings of LSTAG3P (such as MAXGEN, MAXSIZE, operator probabilities,...) were exactly the same with TAG3P. The local search strategy used was stochastic hill-climbing. Lamarckian inheritance was used. For each setting of each system, some 100 runs were conducted, making the total number of runs for this experiment 1200.

The following table shows the proportion of success for all systems. Once again, GP results are also given. Figures 6.13 depicts the cumulative frequencies of all systems and settings on four test problems.

Table 6.10. Proportion of success on the test problems.

Problem	GP	TAG3P	LSTAG3P100	LSTAG3P50	LSTAG3P20
			{200}	{100}	{50}
QUINTIC	65%	78%	90%	85%	78%
SEXTIC	51%	70%	85%	85%	74%
TRIGO	36%	36%	62%	58%	53%
TWOBOX	6%	23%	38%	41%	29%

The results clearly show, that for all problems, LSTAG3P performed better than TAG3P (and GP). For each problem, there are at least two settings, in which the superiority of LSTAG3P over TAG3P and GP is statistically significant (using

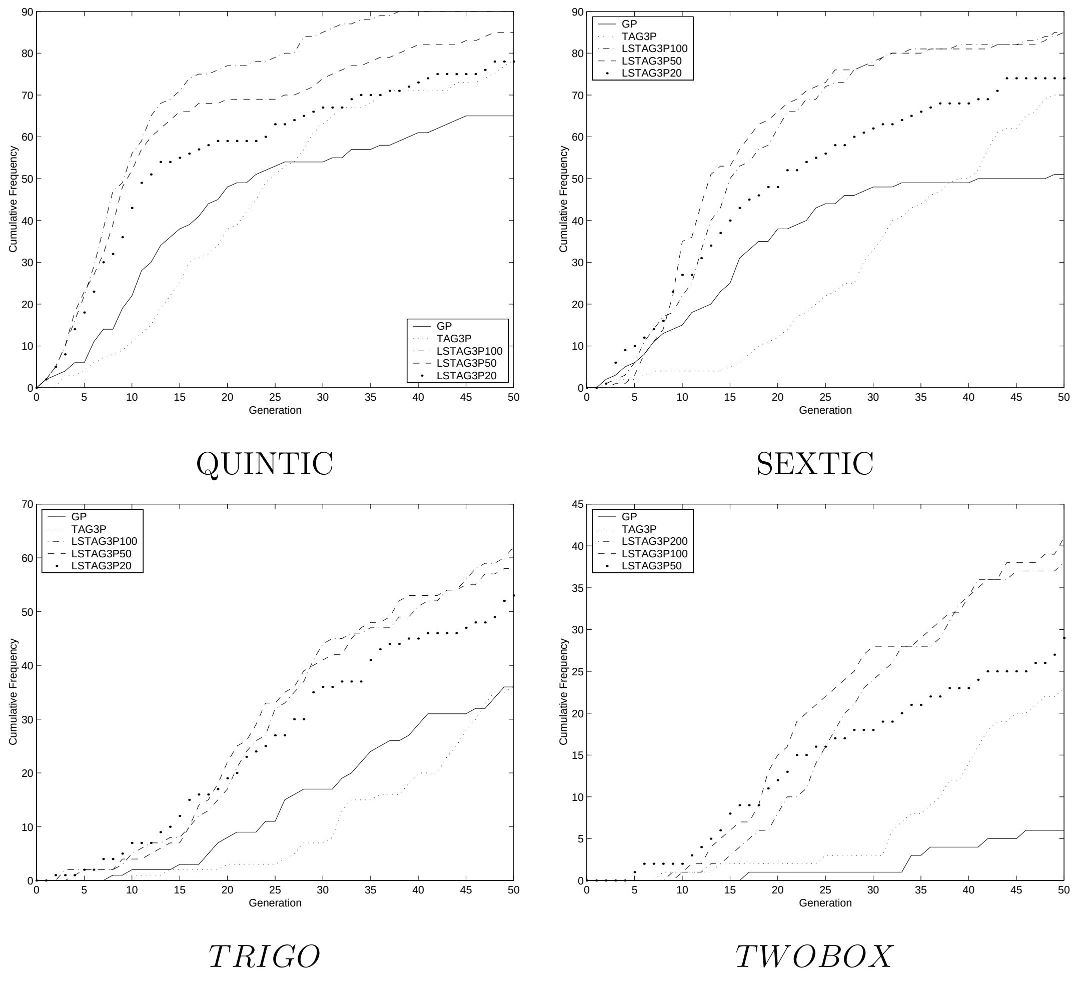


Figure 6.13: Cumulative Frequencies for four problems

one-tailed statistical test of the difference between two random binomial variables with $\alpha = 0.01$). This means that for all problems, whose solutions have some repeated pattern, using relocation as a meta-local search operator helps TAG3P to improve its performance. Moreover, as in the case of insertion and deletion, LSTAG3P could solve the problem with exceedingly small population sizes.

Experiment 3

The purpose of experiment 3 is similar to that of the previous section, where we want to confirm the claim that the superior performance of LSTAG3P comes from the power of relocation, not from shifting towards the exploitation end in evolutionary search by reducing the population size. In other words, the purpose of the comparison here is to compare between TAG3P using the relocation operator and TAG3P, GP having more exploitative power by using small population sizes.

Consequently, GP and TAG3P were run with the same population sizes as in LSTAG3P. However, for each TAG3P (GP) run in this experiment, the number of generations was correspondingly increased so that the maximal number of fitness evaluation is the same with LSTAG3P. Each system and setting was allocated 100 runs. Therefore, the total number of runs for this experiment was 2400. The four following tables show the proportion of success for all systems and settings.

Table 6.11. Proportion of success on the test problems (POPSIZE=100{200}).

Problem	GP100 {200}	TAG3P100 {200}	LSTAG3P100 {200}
QUINTIC	60%	64%	90%
SEXTIC	49%	59%	85%
TRIGO	15%	18%	62%
TWOBOX	1%	23%	38%

Table 6.12. Proportion of success on the test problems (POPSIZE=50{100}).

Problem	GP50 {100}	TAG3P50 {100}	LSTAG3P50 {100}
QUINTIC	42%	74%	85%
SEXTIC	32%	55%	85%
TRIGO	13%	16%	58%
TWOBOX	0%	14%	41%

Table 6.13. Proportion of success on the test problems (POPSIZE=20{50}).

Problem	GP20 {50}	TAG3P20 {50}	LSTAG3P20 {50}
QUINTIC	48%	60%	78%
SEXTIC	45%	65%	74%
TRIGO	9%	19%	53%
TWOBOX	0%	14%	29%

The results in the above tables 6.11, 6.12, and 6.13 show the superiority of LSTAG3P over TAG3P and GP using the same population sizes. It shows that the superior performance of LSTAG3P over TAG3P (and GP) actually comes from the power of relocation when used as a meta-local search operator, not from the effect of reduced population sizes.

Experiment 4

Similar to the previous subsection on insertion/deletion, GP also was run with local search using subtree mutation as the local search operator (LSGP). The population sizes were 100 (200 for TWOBOX problem), 50 (100 for TWO BOX problem), and 20 (50 for TWOBOX problem). The purpose is to separate out the effect of using relocation as the local search operator and the local search strategy itself. Each setting of LSGP was allocated 100 runs, which made the total number of runs for this experiment as 1200 runs. Table 6.14 shows the results for LSGP and LSTAG3P (to keep the table within the page margin, LSGP100 (50, 20) is shorten as GP100 (50, 20), and LSTAG3P100 (50, 20) is shorten as TAG100 (50, 20)).

Table 6.14. Proportion of success on the test problems.

Problem	GP100 {200}	TAG100 {200}	GP50 {100}	TAG50 {100}	GP20 {50}	TAG20 {50}
QUINTIC	62%	90%	61%	85%	48%	78%
SEXTIC	66%	85%	68%	85%	60%	74%
TRIGO	19%	62%	16%	58%	20%	53%
TWOBOX	12%	38%	18%	41%	16%	29%

The results in Tables 6.14 show that LSTAG3P outperformed LSGP on all problem instances in all setting. Thus, it leads to the conclusion that the high performance of LSTAG3P did indeed come from the use of relocation as local search operator but not from the local search strategy it self.

6.2.3 Relocation Operator Conclusion

As in the case of the insertion and deletion operators, the same conclusion on the usefulness of relocation is reached. For relevant problems, such as those that have solutions or partial solutions with repeated patterns, relocation can help TAG3P to improve its performance by rearranging the subcodes within each individual of the population. Moreover, when acting as a generic-local search operator, it helps TAG3P solve problems with very small population sizes.

6.3 Duplication Operator

The second kind of genetic transposition in genome evolution is genetic duplication, whereby a transposon makes copies of itself in other places [Drl1984, Rid1996]. Ohno ([Ohn1970]) even went further, to suggest that genetic duplication is the major force of evolution.

In the field of evolutionary algorithms (EAs), Schwefel ([Shw1968]) was probably the first researcher to use gene duplication in solving real-world problems in industry. In [Hol1975], the operation of gene duplication was also proposed in order to raise the power of EAs. In general, the use of gene duplication has the potential to multiply useful building blocks within individuals, and later the copied building blocks can be subjected to change at their new locations by subsequent gene operations.

In the field of genetic programming, gene duplication has been studied in several forms. In [Koz1995a, Koz1995b, Koz1995c, Koz1996], gene duplication was implemented by copying automatically defined function branches in multi-part programs. Hayes ([Hay1998a, Hay1998b]) also implemented a kind of gene duplication for evolving collective behaviours whereby codes were exchanged between individuals in the population. However, the implementation of general purpose gene duplication operators acting directly on standard GP expression trees (or the executing branch of GP with ADFs) has not been done. Although, in [Hay1996a], a gene duplication operation was defined for GP expression trees and was shown to be useful, the implementation is ad-hoc and problem dependent. We believe that this difficulty comes from the fixed-arity property of the GP expression tree representation.

For GEP, the linear representation facilitates the design of genetic duplication [Fer2001, Fer2002a]. However, just as in the case of relocation, the duplication of any trunk of genes (subcode) in GEP can potentially affect the positions as well as the expressiveness (i.e. coding or non-coding) of many other genes, not just at the source and destination positions. Therefore, it creates a global random side effect on the phenotype. Nevertheless, it was shown to be useful for GEP in some cases [Fer2002a].

In grammar guided genetic programming (GGGP), where the structure of the programs is constrained by grammar rules, it is more difficult than in GP to implement genetic relocation on the genotypic level (derivation trees of the grammars) because of the rule-based nature of the formalism. GE is once again an exception, as with the linear structure of the genotype, it is easy to implement genetic duplication [Rya1998a]. However, as with GEP, since the GE genotype-phenotype map does not posses locality property, the duplication of trunks of genes (subcodes) in GE might completely change the meaning (if there is more than one non-terminal symbol in the grammar) and the expressiveness of the genes following the destination position. The meaning of the duplicated genes might also change vastly, depending on the context before the destination position. Therefore, as with GEP, it creates a global random side effect on the phenotype. To date, there has not been any concrete results in the use of genetic relocation in GE of which we are aware of.

6.3.1 Description of Duplication Operator

With the non-fixed arity property of TAG-derivation trees, gene duplication on the TAG-based representation can be implemented in a simple and natural way. Furthermore, thanks to the locality property of the TAG-based representation, copying a subcode affects neither the meaning of the subcode itself, nor other parts of the tree. Therefore, it is hoped that duplication will help to multiply correct building blocks within each individual.

The duplication operator on an individual, a G_{lex} derivation tree t, is implemented as follows.

```
1)  Choose a subtree $v$ (subcode) in $t$ as random.
    Suppose that $v$ rooted with node $l$.
2)  Find the set $V$ defined as
    $V$={nodes $n$ in ($t$)/$n$ has at least
    one NULL-adjunction that $l$ can adjoin into}
3)  IF $V \neq \emptyset$ THEN
     Choose $m$ uniformly randomly from $V$.
     Choose the address $a$ from all the adjoining addresses in $m$
     that $l$ can adjoin into.
     Create a copy of $v$ from $t$.
     Attach the copy of $v$ to $t$ at $m$ with adjoining address $a$.
     Exit.
4   Loop from 1 to 3 until a maximal number of attempts is exceeded.
```

Figure 6.14 depicts how duplication operator works. It is noted that the duplication operator does change the size of the individual.

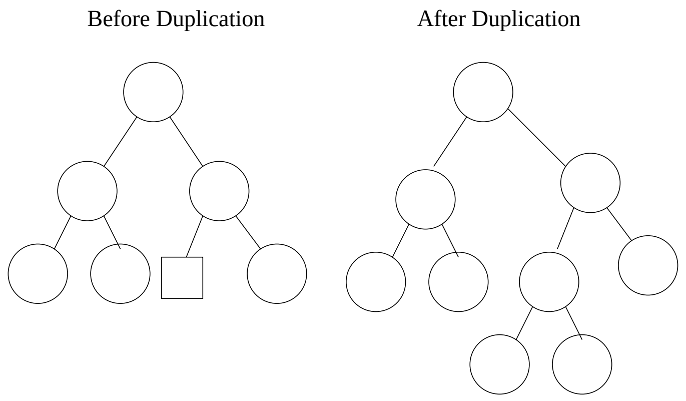


Figure 6.14: Duplication Operator

6.3.2 Experiments

Although gene duplication is bio-inspired and potentially useful for genetic programming in general, and TAG3P in particular, a direct application of the duplication operator might not work. The duplication operator increases the size of the individuals that it is applied to, so, without any size control strategy, code size will potentially bloat very quickly. Some preliminary runs support this assumption. Therefore, in the experiments in this chapter, we used duplication with its counterpart, namely, the truncation operator described in chapter 4.

As in the cases of the insertion, deletion and relocation operators, the roles of duplication and truncation both as mutation operators and as a dual generic-local search operator were investigated.

Test Problems

The problem for experimentation in this section is simple symbolic regression. The details of the problem description and its grammars were given in the previous chapter. The target functions used here are a family of 6 polynomial functions of increasing structural complexity: $X^4+X^3+X^2+X$ (X4), $X^5+X^4+X^3+X^2+X$ (X5),$X^6+X^5+X^4+X^3+X^2+X$ (X6), $X^7+X^6+X^5+X^4+X^3+X^2+X$ (X7), $X^8+X^7+X^6+X^5+X^4+X^3+X^2+X$ (X8), $X^9+X^8+X^7+X^6+X^5+X^4+X^3+X^2+X$ (X9). It is noted that there is a great self-similarity (repeated patterns) in the structures of the above polynomials, and on the interval of interest ([-1..+1]), the polynomials with higher degree can be well approximated by those with lower degree. Therefore, the multiplication of building blocks by the copying of useful subcodes within a polynomial lower degree might help to accumulate more partial polynomial parts (such as $X+X*$) to thus become (or approximate well) a higher degree polynomial.

Experiment Setup

The base runs of TAG3P (TAG3P) used the same setting as in the previous chapter (such as POPSIZE=500, MAXGEN=51, MAXSIZE=40, crossover rate=0.9, mutation rate=0.1,..). The same number of runs with similar settings as in the previous chapter for GP were also allocated and the results are also given here as a reference.

Experiment 1

In the first experiment, as in the previous section, the duplication and truncation operators are used as mutation operators in TAG3P(TAG3PM), and the results were compared with standard GP and TAG3P (TAG3P) using subtree mutation. To separate out the effect of using duplication as the mutation operator from the pure power of subtree crossover in TAG3P, a set of runs was allocated to TAG3P (TAGCROSS) using subtree crossover as the sole genetic operator. For each of the system and problem instances, 100 runs were allocated, making total number of runs to be 2400.

Table 6.15 below shows the proportion of success on all six problem instances. Figure 6.15 shows the cumulative frequencies of GP, TAG3P, TAGCROSS, and TAG3PM on those problem instances.

Table 6.15. Proportion of success on the test problems.

Problem	GP	TAG3P	TAGCROSS	TAG3PM
X4	9%	93%	93%	93%
X5	3%	82%	87%	90%
X6	1%	43%	61%	64%
X7	2%	48%	43%	53%
X8	2%	12%	27%	29%
X9	0%	22%	22%	20%

The results indicate, that on the problem instances tried, TAG3PM works better than TAG3P (and GP) with a statistical significant ($\alpha = 0.05$) for X5, X6, and X8, while its performance was just slightly better than (not statistically significant) or about the same with TAG3P on the others. Moreover, for almost al problem instances above (except X9), TAG3PM's performance was better than TAGCROSS but not statistically significantly. This means that for the problem instances tried, duplication and truncation are better mutation operators than the more standard subtree mutation in TAG3P. However, there is not enough statistical evidence to draw a conclusion that the better performance of TAG3PM is due to the power of duplication and truncation, as mutation operators as its performance was only slightly better than TAG3P using crossover as the sole genetic operator (TAGCROSS). In addition, it is noted that the performance of TAG3PM deteriorates rather quickly when the structural complexity of the target functions was scaled up (from X4 to X9).

Experiment 2

The second experiment is similar to those in the previous two sections, whereby duplication and truncation were used as a dual generic-local search operator in combination with genetic search in TAG3P using subtree crossover and subtree mutation. The results were compared to TAG3P (using full population size - 500). To compensate for the fitness evaluations taken for local search, the population sizes were set as 50 (LSTAG3P50) and 10 (LSTAG3P10). Consequently,

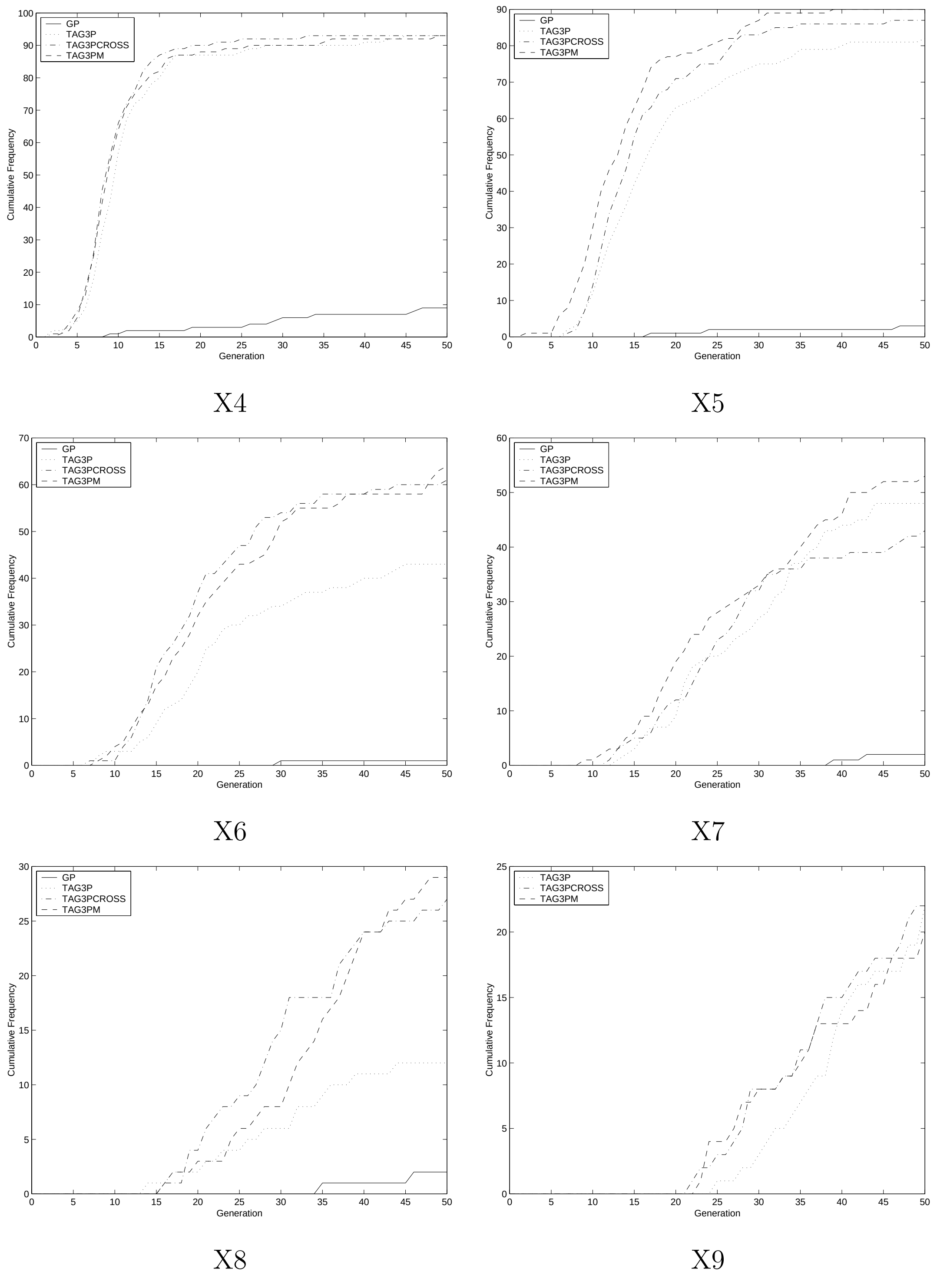


Figure 6.15: Cumulative frequencies for X4-X9

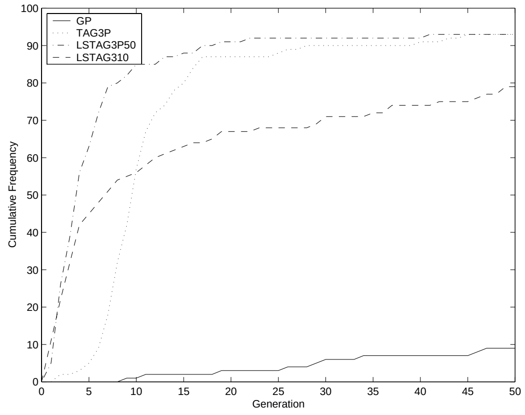
the numbers of local search steps were 10 and 50 respectively. Therefore, the maximal number of fitness evaluation is just the same as in TAG3P (using POP-SIZE=500). Other parameter settings of TAG3P50 and TAG3P10 are similar to those for TAG3P (such as POPSIZE, MAXGEN, MAXSIZE,...). The local search strategy was stochastic hill-climbing, and Lamarckian inheritance was used. On each problem instance, each system was allocated 100 runs, which made the total number of runs in this experiment as 1200.

Table 6.16 below shows the proportion of success on all six problem instances. Figures 6.16 shows the cumulative frequencies of GP, TAG3P, LSTAG3P50, and LSTAG3P10 on those problem instances.

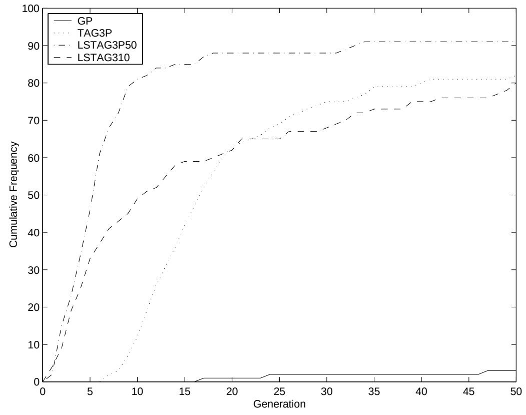
Table 6.16. Proportion of success on the test problems.

Problem	GP	TAG3P	LSTAG3P50	LSTAG3P10
X4	9%	93%	93%	79%
X5	3%	82%	91%	80%
X6	1%	43%	85%	69%
X7	2%	48%	74%	71%
X8	2%	12%	59%	61%
X9	0%	22%	67%	60%

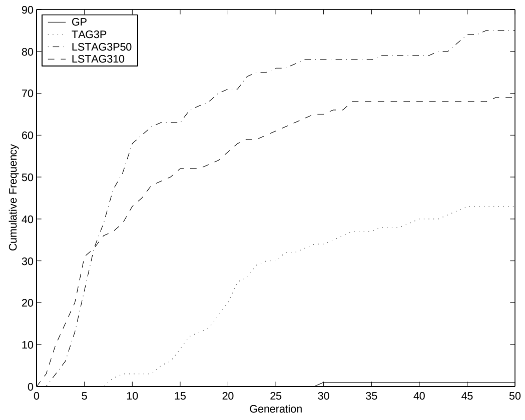
From the results in table 6.16 and figure 6.16, it is obvious that, on the all problem instances tried, LSTAG3P50 outperformed TAG3P significantly, especially when moving to target functions with higher complexity in their structure (i.e. with high repeated and self-similarity patterns); for TAG3P10 the, same conclusion holds except the first two functions. This indicates that duplication and truncation, when being used as a dual generic-local search operator in combination with genetic search, can help TAG3P to significantly improve the performance. Moreover, this performance is still very reliable as the structural complexity of the target function is scaled up.



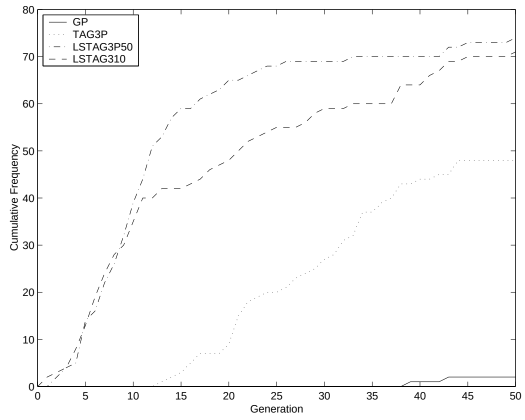
X4



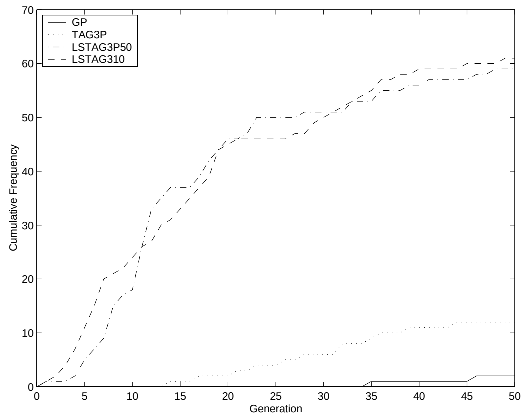
X5



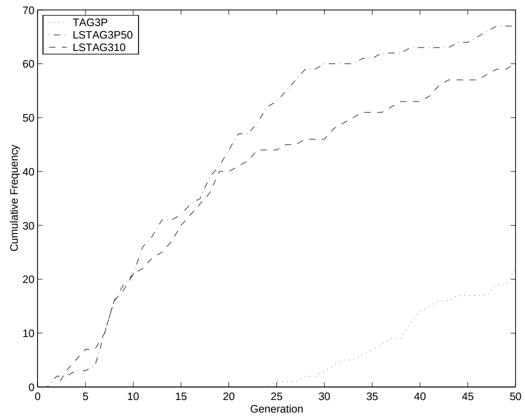
X6



X7



X8



X9

Figure 6.16: Cumulative frequencies for X4-X9

Experiment 3

Similarly, as for the relocation, insertion/deletion operators, the purpose of the third experiment in this section is to show that the superior performance of LSTAG3P did come from the power of duplication/truncation, and not from favouring exploitation by reducing the population sizes. Therefore the same population sizes (50 and 10) were used for TAG3P (TAG3P50, TAG3P10 - using subtree crossover and subtree mutation), while the maximal numbers of generation (MAXGEN) was made bigger correspondingly (511 and 2551). Tables 6.17 and 6.18 shows the results of TAG3P50 and TAG3P10 compared with LSTAG3P50, LSTAG3P10.

Table 6.17. Proportion of success on the test problems (POPSIZE=50).

Problem	GP50	TAG3P50	LSTAG3P50
X4	12%	70%	93%
X5	6%	66%	91%
X6	0%	48%	85%
X7	2%	47%	74%
X8	2%	32%	59%
X9	3%	29%	67%

Table 6.18. Proportion of success on the test problems (POPSIZE=10).

Problem	GP10	TAG3P10	LSTAG3P10
X4	4%	51%	79%
X5	2%	46%	80%
X6	4%	36%	69%
X7	4%	26%	71%
X8	2%	27%	61%
X9	2%	28%	60%

These results clearly show that LSTAG3P outperformed GP50 (10) and TAG3P50 (10) statistically significantly (with $\alpha = 0.01$). Therefore, the superiority of LSTAG3P does indeed come from the power of duplication/truncation operator, and not from the exploitation bias caused by using small population sizes. It is noted however, the claim here for duplication operator, as indicated from the experiments, might be only applicable for problems with highly repeated patterns like the family of polynomials used in this part.

Experiment 4

The purpose of experiment 4 is similar to those in the previous subsections, which is to determine whether the high performance of LSTAG3P did really come from the power of duplication/truncation operators when being used as the local search operator or from the local search strategy itself. GP again was run with local search using subtree mutation as the local search operators (LSGP). The number of runs for each setting was 100, which made the total number of runs as 1200. Table 6.19 shows the results.

Table 6.19. Proportion of success on the test problems.

Problem	LSGP50	LSTAG3P50	LSGP10	LSTAG3P10
X4	6%	93%	7%	79%
X5	3%	91%	1%	80%
X6	2%	85%	0%	69%
X7	0%	74%	1%	71%
X8	4%	59%	0%	61%
X9	0%	67%	0%	60%

The results clearly show that LSTAG3P outperformed LSGP with wide margin. Therefore it backs the conclusion that the high performance of LSTAG3P did come from the power of duplication/truncation when being used as the local search operator but not from the use of local search strategy itself.

6.3.3 Duplication Operator Conclusion

The experiments in this section show that duplication, when using with truncation, can play two roles in TAG3P. Firstly, duplication and truncation are better mutation operators than the more standard subtree mutation on problems where duplication of subcodes is likely to cause the fitness to improve, such as for the test problem in this section, where the task was to learn a family of polynomials of increasing degree. However, when being used as mutation operators, they did not help TAG3P to solve the problems reliably when the structural complexity of the target function was increased. On the contrary, when duplication and truncation played their second roles as a dual generic-local search operator, they not only improved the performance of TAG3P, but also helped TAG3P to cope well when the problem complexity was scaled up. Moreover, they helped TAG3P to solve problems with very small population sizes.

6.4 Conclusion and Future Work

Since it was developed, genetic programming (GP) seems to have had a very limited number of operators that operate directly on its expression trees. There seems to be a consensus that the reason GP does not have the same benefits of employing various operators as its predecessor, GAs, is its tree-based representation. In grammar guided genetic programming (GGGP) the problem is even more difficult to solve, as the derivation tree representation in GGGP is not only tree-based, but also grammar(usually rule based) constrained.

There have been a number of successful attempts to linearise standard GP and GGGP representations. Those attempts resulted in a number of well-known GP and GGGP systems, notably, GEP and GE. With the linearity in genotype structure (like in GAs), it is easy for those systems to implement various bio-inspired operators found in literature. However we argue that, if the genotype-to-phenotype map achieved by the linearisation does not possess the locality (causality) property, which certainly is the case for GEP and GE, it might be difficult to find the context where the operators will work, since the non-locality property in the genotype-phenotype mapping may create a hard-to-control global random side effect on the phenotype (e.g the two examples on the noncausality of GE and GEP in chapter 2).

In this chapter, we have shown that the object-based nature and the non-fixed arity property of TAG derivation trees help the TAG-based representation to overcome the difficulty of designing various operators in standard GP and GGGP. Moreover, since TAG-based representation has the locality property, it is easier to find the context where those operators are potentially useful.

Three groups of bio-inspired operators have been investigated in this chapter, namely insertion/deletion, relocation, and duplication/truncation. The results showed that on relevant problems, they worked well as mutation operators, but did not cope well as problem complexity was scaled up. However, when they were used as generic-local search operators in combination with genetic search, they helped TAG3P to solve the problems well, even with increasing problem complexity. Moreover, they helped TAG3P to achieve that goal with exceedingly small population sizes.

The local search strategy used in this chapter (hill-climbing) is very naive. In the future, we will extend the study towards using some more intelligent local search strategies, such as simulated annealing ([Aar1989]) and Tabu search ([Glo1997]).

Chapter 7

A Schema Theory for TAG3P

Since John Holland proposed his schema theorem in the mid-1970s [Hol1975, Gol1989], schemata are often used to explain why GAs work. Schemata are seen as similarity templates representing entire groups of chromosomes in the population [Lan2002]. Holland's schema theorem describes how the schemata propagate from generation to generation under the influence of selection, crossover and mutation. Although there have been a number of criticisms among GA researchers over the schema theorem [Alt1995, Kar1995, Wol1996, Vos1999], it is still a simple, easy-to-understand, and concise description of the way GAs conduct their search. The problem, as stated in [Rad1997], is not the schema theorem but rather its over-interpretation. In any case, a schema theorem is usually the first stepping stone in understanding the behavior of GAs. A brief introduction to the concepts and terminology commonly used when describing the schema theorem in GAs is given in Appendix B.

In this chapter, we first briefly survey the schema theorems in genetic programming and grammar guided genetic programming. Then, we define the concept of a schema on TAG-based representation, and show that it unifies the three important aspects of a schema on a syntactically constraint domain. Finally, we present a simple schema theorem, estimating the expected lower bound for the propagation of a schema in a TAG3P system using fitness-proportionate selection, subtree crossover, and subtree mutation.

7.1 Schema Theory in GP

In GP, the structure of program chromosomes is usually more complicated than the linear (and binary) representation in GAs. Consequently, as noted in [ORe1995], the extension to GP of some concepts in GA schemata such as "order" and "defining length" is not straightforward. O'Reilly and colleagues suggested that different representations will inevitably lead to different concepts of schemata [ORe1995]. Consequently, there have been a number of different attempts to define a schema and subsequently a schema theorem in GP.

In the early days of GP, Koza ([Koz92]), Altenberg ([Alt1994]), and O'Reilly and Oppacher ([ORe1995]) were the first researchers to try to define a schema for GP systems using expression tree representation. The latter two papers also gave schema theorems based on their concepts of GP schemata. A detailed survey of these schema concepts was given in [Lan2002]. However, as pointed out in [Lan2002], their concepts of schemata as components of programs (usually as subtrees or subtree fragments) although reflecting the component aspect of schemata in GAs, do not resemble the subspace aspect of GA schemata. In other words, GA schemata can potentially match components anywhere in the program tree. This problem complicates the computation of schema propagation in the corresponding schema theorems.

For an effective definition, Langdon and Poli ([Lan2002]) argued that a GP schema should be a component of program trees and should also represent a subspace of program trees. In GAs, this is not an issue because of their fixed and linear chromosome structure, so that the Holland definition of a schema satisfies both requirements at the same time. It is not obvious how to achieve this for GP with tree-based structure.

In [Ros1997], Rosca was the first person to introduce the concept of positional schemata in GP, by defining them as rooted schemata. Based on Rosca's idea, Poli and Langdon ([Pol1997, Pol1998]) developed a new theory of fixed size and shape schemata in GP, in which they resurrected the concepts of "order" and of "defining length" of a schema. A schema theorem was subsequently proven for this type of GP schema using some rather specialised operators [Pol1997, Pol1998, Lan2002]. Their theory for fixed shape and size schemata is given in Appendix B of this thesis. They subsequently went further to derive a series of exact schema theorems [Lan2002].

7.2 Schema Theory in GGGP

In grammar guided genetic programming, the program trees are generated and delimited by a grammar. Therefore, the concept of schema should relate the schemata to the formalism. Moreover, we argue that, apart from the above two requirements for a schema, it is important that the schema also represent a sub-language of the formalism. If it is a sub-language, all the properties of the formalism language transfer to the schemata, making their syntactical properties uniform with the language of the whole search space (i.e. the tree set of the formalism). If we can define schemata to be at once components of programs, sub-search-spaces, and sub-languages, then we can interpret the propagation of schemata as propagating good templates, sampling good parts of the search space, and/or focusing on relevant sub-languages.

To the best of our knowledge, there has been only one attempt at a schema theory for standard grammar guided genetic programming, namely that proposed in Whigham's ([Whi1995c]) schema theorem. Some recent attempts have been made in GE but no schema theorem for GE has been presented [Rya2004].

Whigham's concept of schemata for CFG-GP (context-free grammar (guided) genetic programming) was based on the notion of partial derivation trees in CFGs. Each schema, in his definition, is a partial derivation tree stemming from some non-terminal symbols of the formalism. However, as pointed out in [Lan2002], Whigham's Schemata are program components but not program sub-search-spaces. This is because his schemata are non-positioned. Consequently, they can occur several times in the one program tree. Figure 7.1 illustrates this situation. Moreover, since a Whigham schema is only a partial derivation tree, the string set (and tree set) of the set of programs containing a given schema does not define a sub-language of the formalism. Nevertheless, this definition of schemata for CFG-GP led to a simple schema theorem [Whi1995c, Whi1996]. Appendix B contains a brief review of the CFG-GP schema theory.

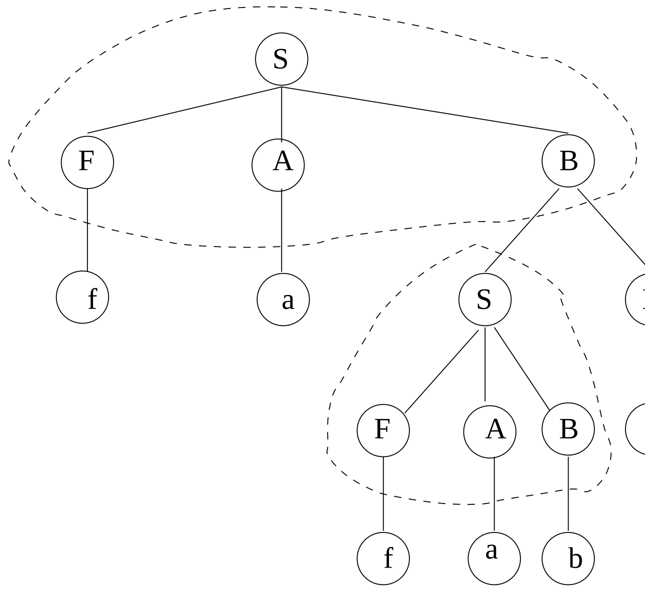


Figure 7.1: An example for Whigham's schema, where a schema can match more than once in an individual

7.3 Schema Definition in TAG3P

The concept of schemata in TAG-based representation is inspired by the rooted schemata of [Ros1997, Pol1997, Pol1998]. Each schema is a template for TAG derivation trees, in which some adjoining addresses are closed or opened for adjunctions. The definition of a schema in TAG-based representation is given as follows:

Definition 1 *A schema in TAG-based representation is a TAG derivation tree. However, all leaf nodes of a schema are labeled with # , where a leaf # means it can be replaced with either a NULL node or a TAG sub-derivation tree, whose* *root is a β-tree that can be adjoined at the address represented by the link between the leaf and its parent node*

Figure 7.2 depicts an example of a schema on TAG-based representation.

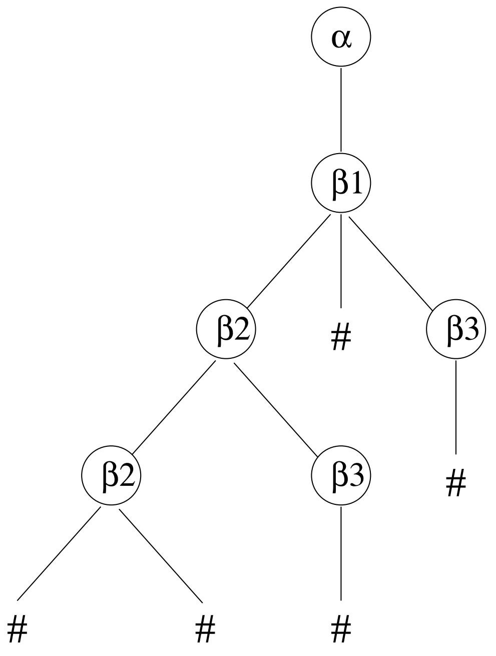


Figure 7.2: An example for a schema on TAG-based representation

Since a schema in TAG-based representation is a rooted, labeled tree, it can occur at most once in each program. Therefore, it satisfies the two requirements for a schema definition proposed by Poli and Langdon. It represents a component of TAG-based representation programs, and it represents a search subspace of TAG derivation trees. It remains to be shown that the set, consisting of all derivation trees in which a given schema occurs, is a sub-language of the TAG used to generate the programs.

Lemma 1. *Let γ be a completed tree derived from some initial α S-trees of a TAG $G_{lex} = \{\sum, N, I, A, S\}$. Then, the tree set consisting of all trees which can be derived from γ by repeated adjunctions and/or substitutions is a sub-language of G_{lex}.*

Proof. It is obvious that any tree γ_1 derived from γ is also derived from some initial S-trees of G_{lex} because γ itself is derived from them. Therefore, the tree set derived from γ using adjunctions and/or substitution must be a subset of the G_{lex} tree language. To see that it is also a TAL, we note that γ is also a completed S-tree, since it is derived from some initial S-trees. Therefore the set can be generated using TAG $G'_{lex} = \{\sum, N, I', A, S\}$, where $\sum, N, A, S$ are as in G_{lex}, and $I' = \gamma$. That completes the proof.

The sub-language aspect of a schema in TAG-based representation is stated in the following theorem:

Theorem 3 *The set of derived trees of the programs initiated by a TAG-based representation schema is a sub-language of the formalism (G_{lex}) used to generate the schema.*

Proof. For any schema H, if all leaves are replaced with NULL, it becomes a derivation tree of G_{lex}. Because of the non-fixed arity property, the derived tree of this derivation tree is a completed tree γ of S-type. Thus, for each G_{lex} derivation tree matching schema H, its derived trees are completed trees derived from γ using adjunction and/or substitutions. By Lemma 1, this tree set is a subset of the G_{lex} tree set, and it is also a TAL. Therefore the set consisting of all derived trees of programs (G_{lex} derivation trees) matching a given schema H constitutes a sub-language of G_{lex}. That completes the proof.

With theorem 3, we can conclude that a schema in TAG-based representation, as defined in Definition 1, unifies all three aspects of a schema on syntactically constrained domains.

7.4 A Schema Theorem for TAG3P

In this section, a simple schema theorem for TAG-based representation schema is derived. The operators used are fitness-proportionate selection, subtree crossover, and subtree mutation. We note that a schema H can only propagate to the next generation if the individuals matching it in the population are selected and transferred to the next generation without being disrupted by genetic operators (of course, some new instances of the schema may be generated through crossover or mutation).

7.4.1 Expected Number of Individuals Matching a Schema after Selection

The aim of a schema theorem is to estimate the rate at which individuals matching a schema are propagated to the next generation. The first stage of this process, selection, is largely independent of representation. To be precise, so long as each individual can match with a schema at most once, and using fitness proportionate selection, the expected number of individuals matching schema H at generation t which are selected for generation $t+1$ is independent of the type of representation. The proof of this proposition can be found in [Hol1975, Gol1989, Lan2002].
Proposition 1. The number (N_s) of individuals matching a schema H being selected with fitness-proportionate selection at generation t is:

$$E[m(H,t+1)] = m(H,t) \times \frac{f(H,t)}{\overline{f}(t)} \tag{7.1}$$

Where $m(H,t)$ is the number of individuals matching schema H at generation t; $f(H,t)$ is the average fitness of individuals matching H at generation t; and $\overline{f}(t)$ is the average fitness of all individuals in the population at generation t.

7.4.2 Schema Disruption due to Genetic Operators

After selection, individuals are transformed using genetic operators. The possibility arises that the children of an individual h matching a schema H may no longer match H. This can happen if and only if the genetic code in h that matches H is destroyed by the genetic operators, usually by choosing operator point(s) in that part, and the new genetic code generated by the operators does not compensate this loss (i.e does not make h match H again). For each $h \in H$, we can formulate an upper bound for the probability that its children do not match schema H as follows:

Proposition 2. The upper bound for the disruptive probability of schema H on an individual h after the application of a genetic operator on h is $\frac{o(H)}{n(h)}$. Where $o(H)$ and $n(H)$ are the numbers of non-# nodes in H and h respectively.

Proof. Since the number of non-#nodes in an individual h matching H is $n(h)$, there are $n(h)$ points for genetic operators to act on. The disruption of H on the children of h (after genetic operations) happens if the chosen points are in the matching region of h and H, which has $o(H)$ nodes. Therefore the probability for a point in the matching region to be chosen is $\frac{o(H)}{n(h)}$. This is an upper bound because there is a possibility that even when the points for genetic operations are chosen in the match region, the children of h might still match H.

7.4.3 A Schema Theorem for TAG3P

Given the previous two propositions, we can state a simple schema theorem for TAG3P with fitness-proportionate selection, subtree crossover and subtree mutation as genetic operators.

Theorem 4 *The propagation of each schema H in the population satisfies the following bound:*

$$E[m(H,t+1)] \geq m(H,t) \times \frac{f(H,t)}{\overline{f}(t)} \times \left[1-(p_c+p_m)\sum_{h\in H}\frac{o(H)}{n(h)}\frac{f(h)}{\sum_{h\in H} f(h)}\right] \quad (7.2)$$

where

$m(H,t)$ is the number of individuals in the population which match schema H at generation t.

$f(H,t)$ is the mean fitness of all individuals matching schema H.

$\overline{f}(t)$ is the mean fitness of all individuals in the population.

p_m is the subtree mutation probability.

p_c is the subtree crossover probability.

$o(H), n(h)$ are the number of non-# nodes respectively in schema H, and in individual $h \in H$.

$E[m(H, t+1)\ ]$ is the expected number of individuals matching the schema H at generation $t+1$.

$\sum$ The sum is taken over the multiset of all individuals in the population that match schema H (a multiset is a generalisation of a set in which each element may occur multiple times. It is necessary to use multisets here because each individual might appear several times in the population).

Proof of Theorem 4. The lower bound for the number of individuals representing schema H in the next generation $(m(H, t+1)$ is determined by two factors. The first is the expected number of individuals $h \in H$ chosen by fitness-proportionate selection – according to proposition 1, this is $m(H,t) \times \frac{f(H,t}{(f)(t)}$. The second factor is the number of selected individuals where the schema may be disrupted by genetic operators. From proposition 2, an upper bound for the probability of disruption of a schema in an individual h is $\frac{o(H)}{n(H)}$, given that h is selected. The probability that h is selected from the individuals in $m(H,t)$ is $\frac{f(h)}{\sum_{h\in H} f(h)}$. Therefore, an upper bound for the probability that each individual $h \in H$ is disrupted by the genetic operators is $(p_m + p_c)\frac{o(H)}{n(h)}\frac{f(h)}{\sum_{h\in H} f(h)}$. Summing over $h \in H$, we get an overall upper bound for the probability that at least one $h \in H$ is disrupted by the genetic operators. Subtracting from 1 gives us a lower bound on the probability that all $h \in H$ get through to the next generation still matching H. Combining the two factors, the inequality of the theorem is derived. That completes the proof.

From the theorem, it can be seen that if $f(H,t)$ is high compared to $\overline{f}(t)$, and $o(H)$ is small (compared to $n(h)$), the lower bound (the left hand side of the inequality) increases. We note that the smaller the starting γ tree in lemma 1, the larger the sub-language of G_{lex} it produces. That is, when $o(H)$ is small, the sub-language defined by the derived tree sets from the individuals matching H is large. Thus the schema theorem may be restated as: "genetic search on TAG-based representation, using fitness proportionate selection, subtree crossover, and subtree mutation, has a bias toward individuals belonging to schemata that are short in length, high in fitness, and large in terms of sub-language".

7.5 Conclusion

In this chapter, we presented a concept of schemata for TAG-based representation. We showed that these schemata embody all three aspects of schemata on syntactically constrained domains, namely search subspaces, program subcomponents, and formalism sub-languages. A simple theorem was given for estimating a lower bound for the expected number of individuals instantiating a schema in the next generation, based on the number in the current generation (using fitness-proportionate selection, subtree crossover and subtree mutation). The theorem gives a rough estimate of how each schema propagates during the evolutionary process. It helps to explain the behavior of TAG3P in terms of schema sampling, in which the evolutionary process of TAG3P favours schemata that are high in fitness and of low order. From the perspective of sub-language sampling, it favours larger sub-languages.

The schemata defined in this chapter are variable in shape and size. Consequently, the process of calculating the lower bound must take into account the size of every individual in the population. This makes the calculation more complicated and results in a relatively underestimated lower bound. We believe that it should be possible to define schemata with fixed shape and size, similar to the approach of [Lan2002]. By so doing, it may be possible to derive a schema theorem which is more descriptive, simpler to calculate, and subject to a better lower bound. We leave this for future work.

Chapter 8

TAG-based Representation and the Problem of Structural Difficulty in Genetic Programming

In chapter 6, we saw how insertion and deletion can be used to improve TAG3P performance on a number of problems. As mentioned, insertion and deletion simulate the growth and shrinkage of trees in the nature. In other words, insertion and deletion cause minimal change in the structure of an individual genotype (G_{lex} derivation tree). Since, in TAG-based representation, the genotype-to-phenotype map has the locality property, the degree of change caused on an individual phenotype is always small, bounded, and controllable. This statement is proven more formally in the next chapter. Consequently, using the insertion and deletion operators, a search agent can walk "smoothly" through the space of G_{lex} derivation trees, as well as of G_{lex} derived tree structures. We conjecture that this property of insertion and deletion might help to soften the well-known structural difficulty problem in genetic programming.

The structure of this chapter is as follows. In the first section, a brief review of the structural difficulty problem in GP is given. Then we discuss why the space of tree structures might be hard for GP to search. The usefulness of insertion and deletion operators for the problem are also discussed. Finally, we present some experiments and results (of which some parts have been published in [NXH2004c]).

8.1 Structural Difficulty in Genetic Programming

In a series of works ([Dai1997b, Dai1999, Cha2000, Dai2001, Dai2002, Dai2003a, Dai2003b]), Daida et. al. discovered that structure alone can pose great difficulty to standard GP search (using expression tree representation and sub-tree swapping crossover). In particular, they delineated 4 regions of the space of tree structures, as shown in Figure 8.1

Region I is where most of the standard GP solutions lie, i.e. it is easy for GP to find a solution in that area; region II (II_a, II_b) is increasingly difficult for GP to search; region III (III_a, III_b), including fuller trees and thinner trees, is almost impossible for GP to search; and, region IV (IV_a, IV_b) is out of bounds (i.e. infeasible tree structures). Moreover, we note that the bounds of regions II and III are almost linear, meaning that they account for the vast majority of tree structures – even when, as is usual in practical GP, a relatively small search space bound (in terms of size or depth) is used.

To further validate this conclusion, in their latest work [Dai2003b], Daida et al. specified a test problem known as LID. In the LID problem for GP, there is only one function of arity 2 (**join**), and one terminal (**leaf**). The raw fitness of an individual tr depends purely on the structural difference from the target solution. It is defined as follows.

$$Fitness_{raw}(tr) = Metric_{depth} + Metric_{terminal} \tag{8.1}$$

where $Metric_{depth}$ and $Metric_{terminal}$ are defined as follows:

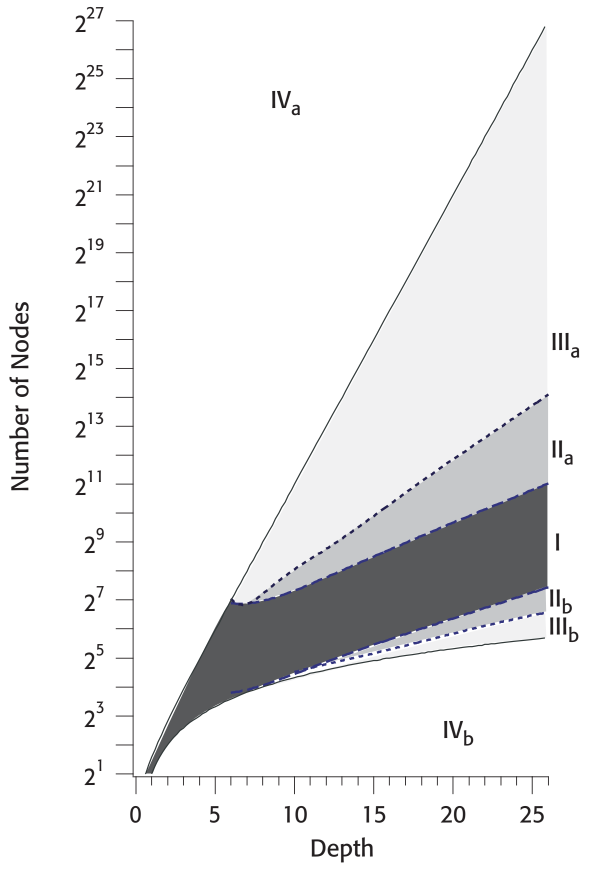


Figure 8.1: Four regions in the space of tree structures. Reprinted with permission from [Dai2003b]

$$Metric_{depth} = W_{depth} \times (1 - \frac{|d_{target} - d_{actual}|}{d_{target}}) \tag{8.2}$$

$$Metric_{terminal} = \begin{cases} W_{terminal} \times (1 - \frac{|t_{target} - t_{actual}|}{t_{target}}) \text{ if } Metric_{depth} = W_{depth} \\ 0 \text{ Otherwise} \end{cases} \tag{8.3}$$

where d_{target}, and t_{target} are the depth and number of leaves of the target solution, d_{actual} and t_{actual} are the depth and number of leaves of the individual (tree) tr. In [Dai2003b], W_{depth} and $W_{terminal}$ are two weighted numbers satisfying $W_{depth} + W_{terminal} = 100$. It is noted that the size s of a tree in the LID problem is related to its t_{target} by the equation: $s = 2 \times t_{target} - 1$.

In [Dai2003b], two families of LID problem instances were used to probe the search space of tree structures, the “horizontal cut” and the “vertical cut”. In the first family, the t_{target} was fixed as 256 and the d_{target} was varied from 8 to 255. In the second, d_{target} was fixed as 15 while t_{target} was varied from 16 to 32768. For a GP system using either size or depth as the chromosome complexity measure, these bounds on size and depth (256 and 15) are quite typical. Figure 8.2 is a simplified version of Figure 8.1 with the positions of the problem instances supperimposed on it.

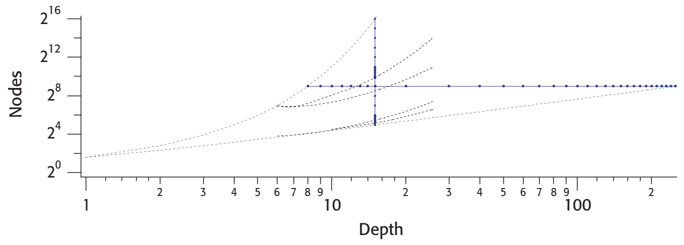


Figure 8.2: The “horizontal and vertical cuts”. Reprinted with permission from [Dai2003b]

Figures 8.3 and 8.4 show the results, based on 90000 runs, of GP on the two families of problem instances. We note that in the Figure 8.4, the x-axis is number of target nodes (s), which is approximately twice the value of t_{target}. The upper parts of the figures show the proportion of successful runs, while the lower show the region to which each problem instance belongs (the cross sign means region I, while the vertical line means regions II and III).

The results in the two figures, surprisingly, show that standard GP, using expression tree representation and sub-tree-swapping crossover, performed extremely poorly on the two families of problem instances, especially for vertical and horizontal cut problems lying in regions II and III. This provides strong evidence that standard GP has considerable difficulty in finding specific structures. Daida et al. [Dai2003b] went further, in showing that the results cannot be fully explained by the sparsity of tree structures in regions II and III (i.e. they are

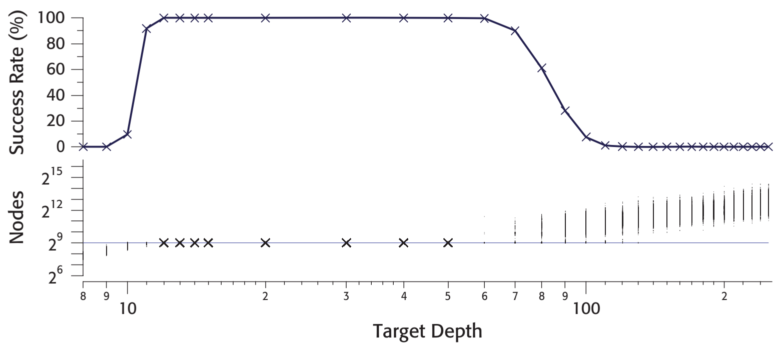


Figure 8.3: Proportion of Success for GP on the "Horizontal cut". Reprinted with permission from [Dai2003b] The lower half of the figure shows which part of the structure space the problem instances belong to. It can be seen that the hard-to-find instances are those in region II, and III

not an equilibrium problem). Their explanation pointed to the expression tree representation itself as the main cause of the structural difficulty.

The structure difficulty problem might play a very important part in the understanding of the behaviour of genetic programming. One of the potential uses of this problem is to explain the code bloat effect. The code bloat effect in GP is a well-documented phenomenon where the code size of evolved programs get larger and larger quickly during the evolutionary process [Bli1994, Ban1998, Sou1999, Lan2002]. The implication from the problem of structural difficulty might be that the reason for bloat is that the target solution has the structure lying in region II or III. Since it is difficult to get solution with structure in those regions, GP might have to accumulate the redundant code to maintain the fitness and as the same time try to find a solution at region I that is equivalent to the target solution in terms of fitness.

Although the Daida results are on standard GP only, the same problem would arise in GGGP, since standard GP is a special case of GGGP [Whi1996]. We conjecture that the problem would be even worse in more syntactically constrained domains.

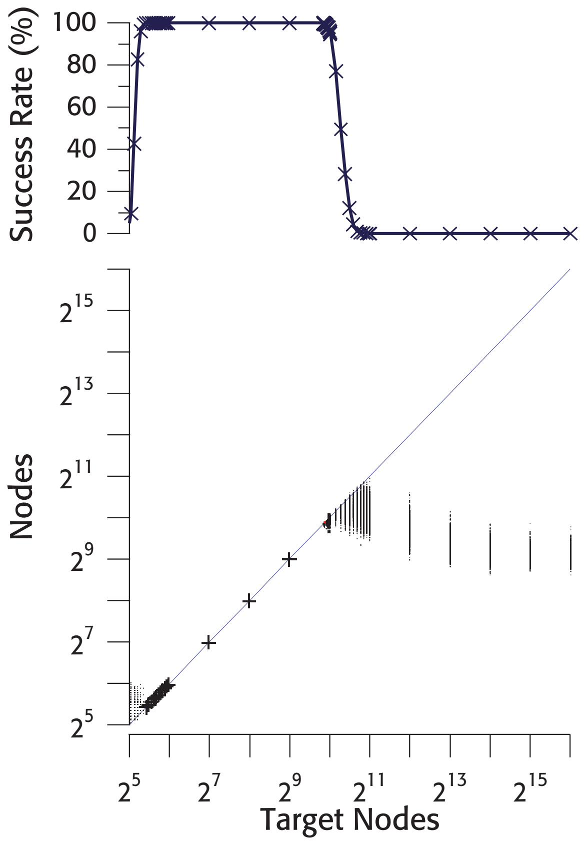


Figure 8.4: Proportion of Success for GP on the "Vertical cut". Reprinted with permission from [Dai2003b]. The lower half of the figure shows which part of the structure space the problem instances belong to. It can be seen that the hard-to-find instances are those in region II, and III

8.2 Insertion and Deletion Operators and the Structural Difficulty in Genetic Programming

Elaborating on Daida's explanation, we conjecture that the problem lies in the structural step size of the structure editing operators in standard GP. In other words, sub-tree crossover and sub-tree mutation, the two main operators in GP, are highly structurally discontinuous - so that the probability of exploring regions 2 and 3 in the space of tree structures can be low despite the presence of selection pressure. We further argue that this discontinuity is a consequence of the fixed-arity property of standard GP representation (that is, each node in the tree has a fixed number of children, determined by its content), in that fixed arity makes it difficult to design operators with controllable step size.

There are at least two ways to validate this hypothesis. One is to analytically analyze the probability of reaching regions 2 and 3 using the particular operator set; the other is to design structure editing operators and show that they can help GP to solve the problem of structural difficulty. We adopt the second course in this thesis.

As pointed out above, our hypothesis is that the problem of structural difficulty in GP can be solved if the GP system supports some efficient structure-editing operators. However, we also argue that, since the expression tree representation in GP is a fixed arity tree, it is difficult to design general purpose structure-editing operators. For GGGP, it is even harder to design such operators, because of the rule constraints at each derivation tree node. On the other hand, TAG-based representation gives us a non-fixed arity tree, facilitating the design of many operators as in chapter 4. Among those operators, insertion and deletion are two structural mutation operators, causing minimal change to the structure of TAG derivation trees. Since the genotype-to-phenotype map has the locality property, the degree of change in the phenotype is also relatively small after applying insertion or deletion. This is formally proven in the next chapter. The results in the next section show that the transformation of representation from GP expression tree to TAG derivation tree, and the use of insertion and deletion, can solve the problem of structural difficulty in GP.

8.3 Experiments

For standard tree-based GP, the LID problem is an extremely difficult search problem. We have argued that this is a result of the operators available, and that the addition of point insertion and deletion operators should greatly ameliorate the difficulty. To demonstrate just how effective the operators are in smoothing the fitness landscape, we use a naive stochastic hill-climbing search (TAG-HILL), which would be readily caught in any local optima remaining in the fitness landscape. Specifically, the algorithm is a (1+1) evolutionary algorithm, i.e. the algorithm uses a population of 1. At each generation, either point insertion or point deletion is selected (with a probability of 0.5). An individual is generated by a random application of the selected operator. If it has better fitness than the parent, it replaces the parent, otherwise it is itself discarded. This loop is repeated for the specified number of steps.

We experimented on the same families of LID problem instances [Dai2003b], namely the horizontal and vertical cuts. The grammar for the LID problem is as follow:

$G = \{N = \{S\}, T = \{Join, Leaf\}, P, \{S\}\}$ where the rule set P is defined as:

$S \rightarrow S\ Join\ S$

$S \rightarrow Leaf$

The corresponding LTAG (using the process from [Jos1997] as in appendix A) is $G_{lex} = \{V = \{S\}, T = \{Join, Leaf\}, I, A)$ where $I \cup A$ is as in Figure 8.5.

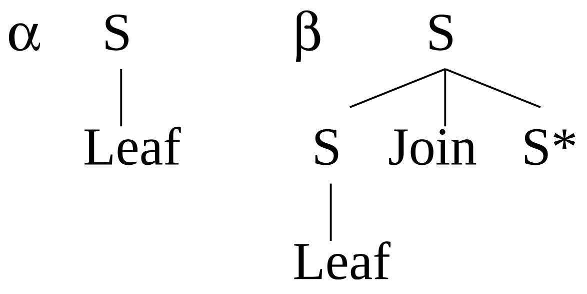


Figure 8.5: Elementary trees of G_{lex} for LID problem

It is noted in [Jos1997] that the mapping from derivation tree of G_{lex} to derived tree of G (and therefore to its expression tree in GP) is one-to-one. However while the GP expression tree for LID is a fixed-arity (binary) tree, the derivation tree in G_{lex} is a non-fixed-arity tree, sometimes called ([Pol1990, Weh1990]) a Catalan tree.

8.3.1 Experiment Setup

In our experiments, the maximal number of steps in TAG-HILL is set to 100000 for each run. This gives the same total number of evaluations as in [Dai2003b], where the size of population and the number of generations are set to 500 and 200 respectively. Consequently, the maximal allowed number of fitness evaluation in TAG-HILL is the same as in the GP experiments in [Dai2003b]. For the horizontal cut, t_{target} was fixed as 256 while d_{target} was varied from 8 to 255. For each varied d_{target}, 100 runs were allocated. Similarly, in the vertical cut, the d_{target} was fixed as 15 while t_{target} was varied from 16 to 32768. For each t_{target} in [16..1024], we carried out 100 runs, while for [1025..32768], we ran 100 trials for each point of the 118 equi-sampled points in that interval. Although the setting of W_{depth} and $W_{terminal}$ do not affect TAG-HILL search, we set them the same as in [Dai2003b], i.e. as 30 and 70 respectively.

In Figures 8.5, the β-tree for the LID problem have three adjoining addresses, namely at the root node, at the foot and at the lower node (labeled S). In the TAG literature, adjunction at the root and foot nodes in a β-tree are usually not used simultaneously, since it creates some redundancy in the mapping between TAG-derivation trees and TAG-derived trees. To investigate the effect of this redundancy of the genotype-to-phenotype mapping on the LID problem, we divided the TAG experiments into two set of runs. In the first set, we used only two adjoining addresses (excluding the root node: 2-ADD), while in the second, all three adjoining addresses were used (3-ADD).

8.3.2 Results and Discussion

The following Figures 8.6 and 8.7 show the proportion of success of TAGHILL (2-ADD and 3-ADD) based on 275200 conducted runs (137600 for each of 2-ADD and 3-ADD).

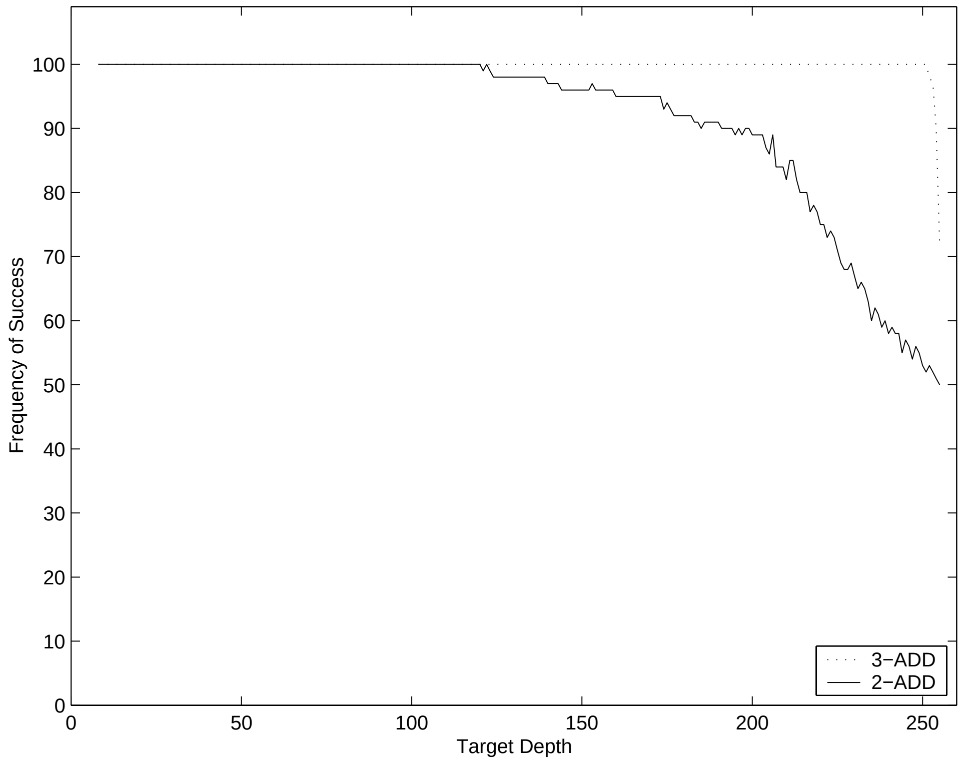


Figure 8.6: Results of TAG-HILL on the "horizontal cut"

In Figure 8.3, GP runs were unreliable in finding solutions for $d_{target} > 70$ or $d_{target} < 10$, and failed to find any solutions for $t_{target} = 8$ or $d_{target} > 100$. By contrast, as shown in Figure 8.6, TAG-HILL 3-ADD could solve the problems with 100% reliability for d_{target} up to nearly 253, and only failed to find solutions with 100% reliability when $d_{target} > 253$. TAG-HILL 2-ADD managed to solve problems with 100% reliability for d_{target} up to 120 and for $d_{target} > 120$ it still found solutions with more than 50% reliability.

For the results on the vertical cut family shown in Figure 8.4, GP runs were unreliable in finding solutions for $t_{target} > 500$, and failed to find any solutions for $t_{target} > 1024$. By contrast, as shown in figure 8.7, TAG-HILL 3-ADD could solve the problems with 100% reliability for t_{target} up to nearly 8000, and only failed to find solutions when $t_{target} > 11000$. TAG-HILL 2-ADD was even more successful, managing to solve problems with 100% reliability for t_{target} up to 17700 and only failing to find solutions when $t_{target} > 21000$.

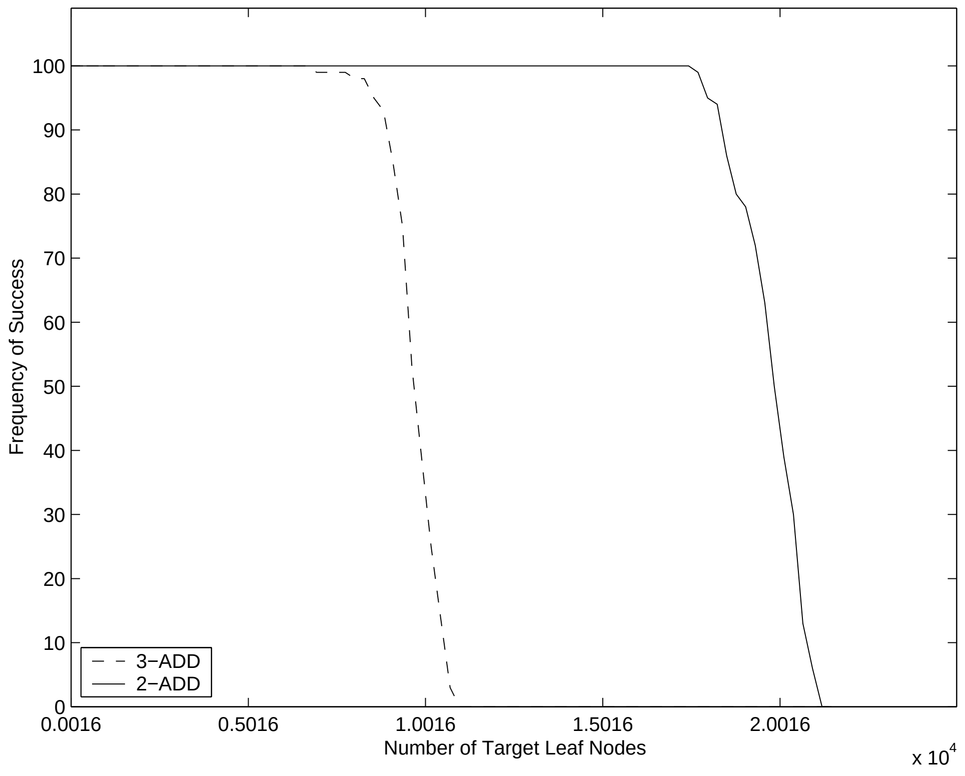
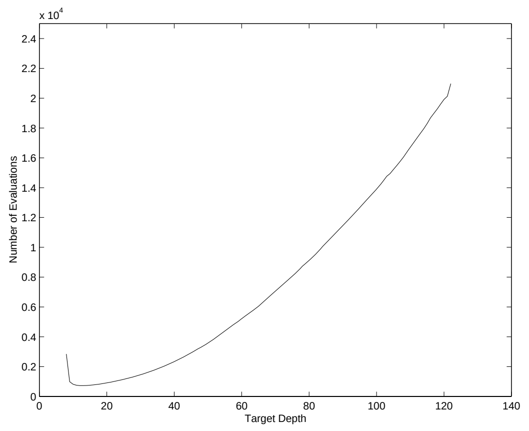
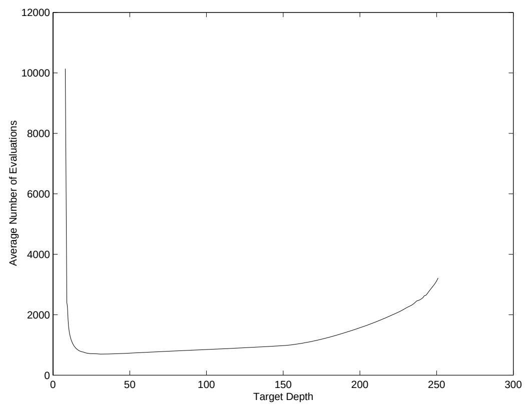


Figure 8.7: Results of TAG-HILL on the "vertical cut"

Figures 8.8, 8.9, show the average number of search steps for TAG-HILL (2-ADD and 3-ADD) to find solutions (for those problem instances where 100% success was achieved). The almost linear scale suggests that, except for some extreme points, the landscape of the two families of LID problem instances is quite smooth for TAG-HILL. This is no trivial matter, since when t_{target} (d_{target}) approach their extreme values, the tree structure become exponentially sparse [Sed1996]. To see just how sparse, take the example of the leftmost point on the horizontal cut, where $t_{target}= 256$ and $d_{target}= 8$. There is only one tree with that combination of t_{target} and d_{target}, out of $\frac{4^{255}}{\sqrt{\pi 255^3}} \approx 2^{497}$ trees in the search space ([Sed1996]).

The results also show that the redundancy in the genotype-to-phenotype map helped TAG-HILL 3-ADD perform much better than TAG-HILL 2-ADD on problems of the "horizontal cut" family, while performing much worse on problems of the "vertical cut" family. We explain this as follows. In 3-ADD, there are more





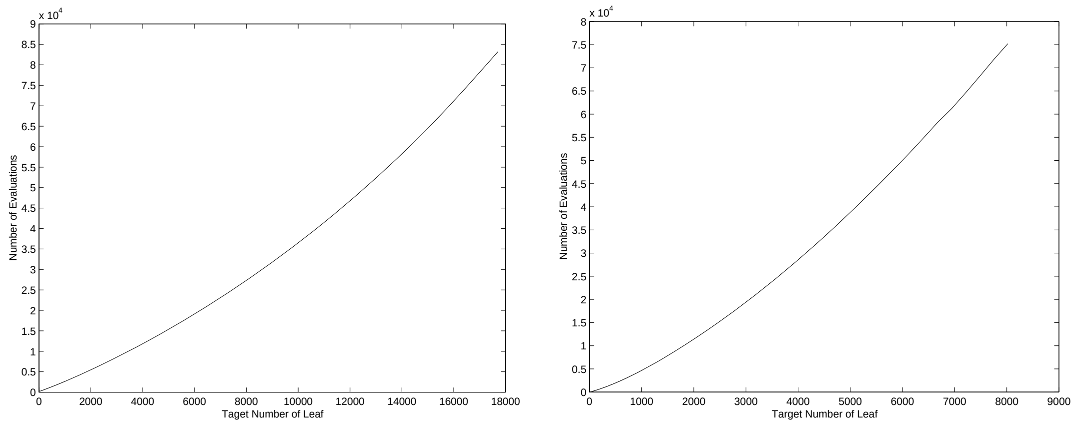
2-ADD 3-ADD

Figure 8.8: Average number of fitness evaluations for the "horizontal cut"

available adjoining addresses than in 2-ADD. When an insertion is to occur, in 2-ADD, many of the adjunction addresses higher in the tree will already have been used, so that the probability of selecting a lower adjunction address (and hence deepening the tree) is increased. In 3-ADD, since more addresses higher in the tree are available, the probability of selecting them is reduced more slowly, with fuller trees being the result.

8.4 Conclusion

We have argued that GP's problem of structural difficulty results from the lack of local structure-editing operators and pointed to the fixed-arity expression tree representation in GP as the primary culprit. Using TAG-based representation, we have removed this fixed-arity limitation. In this representation, we were able to design two local structure-editing operators, namely point insertion and deletion. Applying these operators to Daida's LID problem, we demonstrated that the operators significantly ameliorate the structural difficulty problem in GP. The results also showed that redundancy in the genotype-to-phenotype mapping, created by using redundant adjoining addresses, can affect the efficiency of the



2-ADD 3-ADD

Figure 8.9: Average number of fitness evaluations for the "vertical cut"

operators in finding solutions in tree structure space.

In this chapter, we used the rather naive hill-climbing search strategy. Further work will investigate the behaviour of other adaptive search strategies. On a theoretical level, we are endeavouring to derive a formula to predict the convergence time for TAG-HILL on the LID problem. We are also attempting to analyze the behavior of GP using sub-tree crossover and/or sub-tree mutation on the LID problem, to provide an analytical measure of the structural difficulty problem, and to validate our hypothesis that the difficulties are caused by the structural discontinuity of the standard sub-tree operators in GP.

Chapter 9

Fitness Landscape Study on Syntactically Constrained Domains

In chapters 4 and 6, we have shown that, thanks to the non-fixed arity property of TAG derivation trees, it is possible to design operators such as insertion, deletion, and point replacement on TAG-based representations. These operators make bounded changes in the genotype space (G_{lex} derivation trees). Since TAG-based representation possesses the locality property (as argued in chapter 3), the changes in the phenotype space (G_{lex} derived trees) are also bounded. In this chapter, we argue that these properties of TAG-based representation are particularly useful in helping to understand problem difficulty, by investigating the characteristics of problem fitness landscapes.

The chapter starts with a brief description of the role of fitness landscape study in the field of evolutionary algorithms (EAs). It is followed by a discussion of the difficulties in fitness landscape study in genetic programming, and especially in grammar-guided genetic programming. Next comes an examination of how those issues relate to fitness landscape studies using TAG-based representation. Finally, two fitness landscape studies in TAG-based representation are presented. The purpose of the first is to understand the impact of changing target function in a family of functions, and the second to understand the impact of changing grammars, on the fitness landscape characteristics of some standard problems.

9.1 Fitness Landscape Study in EAs

The idea of fitness landscape was first proposed in [Wri1932]. Since then, it has been used as a tool for analysing evolutionary theories [Kau1993], as well as to model the problem difficulties in evolutionary algorithms [Deb1997, Ree2003]. It uses metaphors from nature such as peaks, hills, valleys, ridges, basins, watersheds etc. to describe the characteristics of a search space that EAs might encounter when exploring it. Even for real-world problems, of generally much higher dimensionality than 2 and 3, the study of such metaphors in the higher dimensional space is still useful. It helps to understand the properties of the search space, such as its ruggedness (i.e. how each function value is correlated with the function values of neighbouring points) and modality (i.e. whether it has one or many local optima). By acquiring and exploiting knowledge about the fitness landscape of the problem, an EA can improve its search performance [Deb1997, Ree2003].

In practice, the concept of fitness landscapes is somewhat vague. As noted in [Ree2003], it has even been misused in the literature to describe fitness functions. Although the fitness function, which is problem dependent, is a defining component of fitness landscape, the topology of the search space must also be defined. In fact, one should not use the term "landscape" before defining the topological structure on the search space [Ree2003]. The topology of a search space in evolutionary computation consists of a genotype representation and a neighbourhood structure defined on it [Ree2003, Vas2000, Vas2003]. The neigbourhood structure is usually defined through a distance metric d on the genotype space [Ree2003]. A genotype l_2 is in the neighborhood of a genotype l_1 if and only if $d(l_1, l_2) < \delta$; when $\delta = 1$ (minimal distance), l_2 is called a neighboring point of l_1. The third component of a fitness landscape is the genetic (search) operators [Ree2003, Vas2000]. In order to investigate the fitness landscape, it is important to be able to "walk" on the genotype space, i.e. to transform one point to a neighbour in the search space. To properly investigate the characteristics of the fitness landscape, the genetic operators need to be consistent with the topological structure of the genotype (search) space [Ree2003]. In other words, the operators should transform a point ideally to its neighbour, or at least within its neighbourhood. In order to ensure that the genetic (search) operators were consistent with the metric defined on the genotype (search) space, some researchers ([Jon1995a, Jon1995b, Vas1999]) defined the metric in terms of the operators. However, the use of genetic operators to define the genotype metric makes the fitness landscape dependent on the operators used, different genetic operators giving different measures of the fitness landscape of a problem [Jon1995b]. On the other hand, the tree-edit distance metric defined in chapter 3 gives a natural distance metric on GP search spaces, independent of particular operators. Gaining an understanding of the fitness landscape with this underlying metric can help to design appropriate operators. Some recent fitness landscape studies in genetic programming have already followed this approach [Van2003a, Van2003b].

9.2 Problems with GP and GGGP in Fitness Landscape Study

In GAs, the task of defining a distance metric on the search space, and designing operators consistent with it, is not particularly challenging. For instance, with linear binary representation and using Hamming distance (the total number of different bits between two strings), one-point mutation is an obvious candidate. In genetic programming, things are more complicated because of the variability in shape and size. As pointed out in chapter 2, the expression tree in GP, and the CFG derivation tree of GGGP, make it difficult to design operators that make small, bounded changes. A similar conclusion was reached in [Lan2002] (page 24):

> ... However, often there is no natural ordering for the structures in fitness landscape. Also, even if there is one, the action of a search operator may be such that it does not respect the natural neighbourhood relationship available for the domain; i.e the operator may allow multi-step jumps...

In [Kin1994b], the autocorrelation function from random walks (described in appendix C) was used to analyze the fitness landscape on expression tree representation for a number of standard problems in GP. The results were somewhat inconclusive, the autocorrelation being blamed as an inaccurate estimator of fitness landscape smoothness. However, the problem might also lie in the disruptiveness of the operators Kinnear used, namely subtree crossover, subtree mutation, and hoist mutation. The operators take long step jumps, ignoring the natural topological structure on expression trees. For instance, in the previous chapter, it seems clear that the disruptiveness of subtree crossover on GP expression trees is an important factor in Daida's problem of structural difficulty in GP.

Recently, [Van2003a, Van2003b] proposed some smaller scale operators on GP expression tree representation, namely the inflate and deflate structural mutation operators. The operators were shown to be consistent with the tree alignment metric (mentioned in chapter 3 of the thesis); and some fitness landscape analysis was conducted using fitness distance correlation techniques. However the operators rely on incrementing and decrementing the arity of GP primitives, so they are inapplicable when all functions have the same arity. This difficulty is all the more pressing in syntactically constrained domains, where it is difficult to imagine how the operators might be made consistent with the grammar constraints.

For linear representations such as GEP, it is possible to use distance metrics and operators from GAs. However, if the representation does not have the locality property (as in GEP), the fitness landscapes study might tell us the characteristics of the genotype (search) space, but provide no guidance to the characteristics of the original (phenotype) space. In other words, although it can help to understand the search difficulty of the problem under the GEP representation transformation, it does not help us to understand the difficulty of the original problem. As noted in [Alb2000, Lan2002], the change of representation using a genotype-phenotype mapping can change the fitness landscape significantly. [Rot2002] showed that this only applies to genotype-phenotype mappings lacking the locality property. Mappings satisfying the locality property do not change the fitness landscape much, since the neighborhood structure is preserved (i.e. if l_1 is in a neighborhood of l_2 then $f(l_1)$ is in a neighborhood of $f(l_2)$, where f is the transformation from genotypes to phenotypes). It is a direct consequence of "small changes in genotypes resulting in small changes in phenotypes" (locality property) [Pal1994b].

In the field of grammar guided genetic programming, where the problem domains are further constrained by grammar rules, it is even harder to design operators that make small and bounded changes. Whigham proposed the study of fitness landscape ([Whi1996] chapter 7) on syntactically constrained domains, so as to understand the impact of biasing the language of programs on the landscape. To the best of our knowledge, there has been no subsequent work on fitness landscape studies for syntactically constrained domains. For grammar guided genetic programming systems with linear representation such as GE, it is easy to define genotype metrics and consistent genetic operators. However if the representation does not possess the locality property (as with GE), the same problems arise as with GEP above. Moreover, in some circumstances, infeasible phenotypes can result from valid genotypes in GE, potentially creating serious difficulties in making a meaningful study of GE fitness landscapes, especially when there is a variety of infeasible phenotypes.

9.3 TAG-based Representation and Fitness Landscape Study

As mentioned in chapter 4 of the thesis, thanks to the non-fixed arity property, it is possible to design operators that make small and bounded changes on the genotype level. Moreover, since the mapping from genotype space to phenotype space has the locality property as proven in chapter 3, if the change on the genotype is small and bounded, it is so on the phenotype space.

The topological structure chosen for the genotype as well as the phenotype spaces is the topology induced by the tree-edit distance described in chapter 3. A complete set of operators respecting that topological structure on the genotype space consists of insertion, deletion, and point replacement described in chapter 4 and 8. This is formulated in the following proposition.

Proposition 9.1. *Insertion, deletion, and point mutation make minimal change on the genotype space (G_{lex} derivation trees) and bounded change on the phenotype space (G_{lex} derived trees).*

Proof. From the definition of tree-edit distance, for any of the three operators o, and for any TAG-derivation tree t, $d(t, o(t)) = 1$, where d is the tree-editing distance. If in theorem 3.2 of chapter 3, we replace d by 1, then it follows that $d(f(t), f(o(t))) \leq M$, where M is the maximal number of nodes in an elementary tree, and f is the genotype to phenotype mapping.

The three operators are complete in that they provide a path from any labeled tree to any other labeled tree [Pol1990]. Using the three operators with equal probability of application, one can walk from any point to any other point both in the genotype space (TAG-derivation tree) and in the phenotype space (since the genotype-phenotype mapping f is onto) of TAG-based representation. This makes these operators ideally suited to study fitness landscape on syntactically constrained domains.

9.4 Experiments

In this section, the triple operators (deletion, insertion, and point replacement) are used to create a random walk for fitness landscape studies on some syntactically constrained domains. At each step of the random walk, one of the three operators is chosen with equal probability (1/3).

The experiments in this section are used both to understand how the fitness landscape changes as the structural complexity of the target functions is varied, and to study the effects of changing grammars on the fitness landscape while retaining a fixed target function.

In standard GA problems, the structural complexity is fixed, so that any difference in difficulty between two problems must arise from their fitness landscapes. In GP, the structural complexity is variable; more complex target functions are harder to generate simply because there is more to learn. So there are two potential sources of differences in problem difficulty for GP, namely structural complexity and ruggedness of the fitness landscape.

In chapter 5, a family of polynomial functions (F_1 to F_4), with increasing structural difficulty, was used to test the robustness of TAG3P compared with GP, and CFG-GP systems. The results showed that as we pass from F_1 to F_4, the problems become increasingly difficult for TAG3P and CFG-GP. Does this performance degradation stem purely from the increasing structural complexity of the target functions, or also from an increase in ruggedness of the fitness landscape? In the first set of experiments, the same grammars (G and G_{lex}) were used for all target functions. For the symbolic regression problem, where the task is to learn the target function from sampled data, the structure is not used in the fitness calculation. However, as indicated by Daida (chapter 8), structure alone could influence problem difficulty. A fitness landscape study of these symbolic regression problems, with fixed grammars but varying the target function from F_1 to F_4, could cast light on the hypothesis in chapter 5 that the increasing difficulty of the symbolic regression problem from F_1 to F_4 stems from the increase in structural complexity of the target functions.

One of the generally-acknowledged advantages of the use of grammars in genetic programming is their ability to set a declarative bias on the search space. By changing the grammar, one can bias the search system towards a particular region of the search space. [Whi1995b, Whi1996] includes a study showing how incorporating increasing knowledge into the grammar can improve the performance of CFG-GP. The test problem used was the 6-multiplexer problem, four grammars (G_1, G_2,G_3, and G_4) being used to define four different levels of declarative bias on the search space. The grammars are as follows ([Whi1996], pages 59-61).

$G_1 = (\sum, N_1, P_1, S)$, $\sum = \{a_0, a_1, d_0, d_1, d_2, d_3\}$, $N = \{S, B\}$, and the rule set P_1 is

P_1:

$S \rightarrow B$

$B \rightarrow if\ B\ B\ B$

$B \rightarrow B\ and\ B$

$B \rightarrow B\ or\ B$

$B \rightarrow not\ B$

$B \rightarrow a_0|a_1|d_0|d_1|d_2|d_3$

$G_2 = (\sum, N_2, P_2, S)$, $\sum$ is the same as in G_1, $N = \{S, B, ADDRESS\}$, and the rule set P_2 is

P_2:

$S \rightarrow if\ ADDRESS\ B\ B$

$ADDRESS \rightarrow a0|a1\ B \rightarrow if\ B\ B\ B$

$B \rightarrow B\ and\ B$

$B \rightarrow B\ or\ B$

$B \rightarrow not\ B$

$B \rightarrow a_0|a_1|d_0|d_1|d_2|d_3$

$G_3 = (\sum, N_3, P_3, S)$, $\sum$ is the same as in G_1, $N = \{S, B, ADDRESS, IFTHEN\}$, and the rule set P_3 is

P_2:

$S \rightarrow if\ ADDRESS\ IFTHEN\ B$

$ADDRESS \rightarrow a0|a1\ IFTHEN \rightarrow if\ B\ B\ B$

$B \rightarrow if\ B\ B\ B$

$B \rightarrow B\ and\ B$

$B \rightarrow B\ or\ B$

$B \rightarrow not\ B$

$B \rightarrow a_0|a_1|d_0|d_1|d_2|d_3$

$G_4 = (\sum, N_3, P_3, S)$, $\sum$ is the same as in G_1, $N = \{S, B, IFA1\}$, and the rule set P_4 is

P_2:

$S \rightarrow if\ a0\ IFA1\ B$

$IFA1 \rightarrow if\ a1\ B\ B$

$B \rightarrow if\ B\ B\ B$

$B \rightarrow B\ and\ B$

$B \rightarrow B\ or\ B$

$B \rightarrow not\ B$

$B \rightarrow a_0|a_1|d_0|d_1|d_2|d_3$

The corresponding TAGs (G_{lex}) derived from them are given in Appendix A of the thesis. From the grammar descriptions, it can be seen that G_1, which is similar to the grammar in chapter 5, is the most general. It generates all Boolean functions that can be formed from the function symbols if, and, or, not and the Boolean variables $a0, a1, d0, d1, d2, d3$. The functions generated by G_2 are a subset of those generated by G_1. G_2 only generates functions starting with an if-then-else statement, with the condition part being an address variable ($a0$ or $a1$). G_3 is biased even further than G_2 since functions are restricted to start with two nested if-then-else statements, and the condition of the first if must be an address variable. G_4 incorporates the strongest bias of all, only generating expressions beginning with two nested if-then-else statements whose condition parts are address variables. As one moves from G_1 to G_4, the search space is biased towards smaller and smaller subspaces. The significant reduction in search space size was argued as the reason for the problem becoming easier as the bias changed from G_1 to G_4 [Whi1995b, Whi1996]. Note that even though the grammar changed, the target function in these experiments was fixed.

However apart from the reduction in search space size, the change of grammar might also change the characteristics of the fitness landscape, through changing the function representation. It is important to decouple these two effects (search space size and fitness landscape). Our second set of experiments conducts a fitness landscape study on the effect of changing the grammars while fixing the target function for this problem.

9.4.1 Experiment Setup

Two experiments were conducted for two problems, symbolic regression and 6-multiplexer. For the first experiment, the grammar is taken from chapter 5, while the target function is varied from F_1 (cubic polynomial function) to F_4 (polynomial function of degree 6 - see chapter 5 for more details). The aim was to determine whether increasing problem difficulty might stem partly from changing fitness landscape, as well as from the increasing structural complexity of the target functions. In the second experiment, the analysis aimed to determine whether decreasing problem difficulty with more specific grammars might arise partly from changing fitness landscape, or whether it was purely a consequence of the reduction in search space size reduction. The problem chosen was the 6-multiplexer and the grammars were G_1 ($G1_{lex}$), G_2 ($G2_{lex}$), G_3 ($G3_{lex}$), and G_4 ($G4_{lex}$).

For each experiment, a random walk of 10,000 steps was conducted using the triple operators (insertion, deletion, and point replacement, with equal probability) to create the walk. All fitness values of individuals encountered during the random walk were recorded. For each problem, 300 random walks were conducted, making a total of 3,000,000 fitness evaluations for each experiment. The data were then analysed using two common techniques from the literature, namely the autocorrelation function and the information content. These techniques are described in Appendix C.

9.4.2 Results and Discussion

Tables 9.1 and 9.2 show the autocorrelation values and the information content of the random walks for the symbolic regression problem – Length is the correlation length; Optima No. is the number of optima encountered during the random walks)

Table 9.1. Autocorrelation analysis for symbolic regression problem.

Length	F_1	F_2	F_3	F_4
1	0.7892 ± 0.0798	0.8002 ± 0.0715	0.7964 ± 0.0703	0.8025 ± 0.0717
2	0.6470 ± 0.1019	0.6620 ± 0.1046	0.6528 ± 0.1025	0.6656 ± 0.1059
3	0.5390 ± 0.1241	0.5561 ± 0.1213	0.5429 ± 0.1216	0.5626 ± 0.1243
4	0.4572 ± 0.1293	0.4750 ± 0.13	0.4593 ± 0.13	0.4819 ± 0.1337
5	0.3924 ± 0.1314	0.4099 ± 0.1332	0.3932 ± 0.1347	0.4157 ± 0.1388
6	0.3409 ± 0.13	0.3570 ± 0.1333	0.3407 ± 0.1354	0.3624 ± 0.14
7	0.2973 ± 0.1280	0.3145 ± 0.1306	0.2976 ± 0.1344	0.3185± 0.1390
8	0.2609 ± 0.1254	0.2794 ± 0.1271	0.2619 ± 0.1314	0.2811 ± 0.1376
9	0.2310 ± 0.1209	0.2488 ± 0.1237	0.2328 ± 0.1282	0.2502 ± 0.1339
10	0.2053 ± 0.11157	0.2224 ± 0.1201	0.2086 ± 0.1237	0.2243 ± 0.1309

Table 9.2. Infomation content analysis for symbolic regression problem.

Function	ϵ	$H(\epsilon)$	$h(\epsilon)$	$M(\epsilon)$	Optima No.
F_1	0	0.5920 ± 0.0137	0.6708 ± 0.0112	0.5071 ± 0.0075	2535 ± 37
F_2		0.5911 ± 0.0142	0.6706 ± 0.0106	0.5064 ± 0.007	2531 ± 35
F_3		0.5909 ± 0.0135	0.6711 ± 0.0108	0.5059 ± 0.0068	2529 ± 33
F_4		0.5912 ± 0.0146	0.6714 ± 0.0105	0.5062 ± 0.0067	2530 ± 33
F_1	1	0.723 ± 0.182	0.6882 ± 0.0111	0.4571 ± 0.0125	2285 ± 63
F_2		0.7235 ± 0.0189	0.6895 ± 0.112	0.4555 ± 0.0126	2276 ± 63
F_3		0.7247 ± 0.0178	0.6891 ± 0.0114	0.455 ± 0.0125	2274 ± 62
F_4		0.7253 ± 0.0183	0.6895 ± 0.0115	0.4546 ± 0.0128	2272 ± 64
F_1	2	0.7588 ± 0.0143	0.6945 ± 0.0118	0.4276 ± 0.0138	2137 ± 69
F_2		0.7571 ± 0.0144	0.6977 ± 0.0119	0.4256 ± 0.0139	2127 ± 70
F_3		0.7580 ± 0.0143	0.6968 ± 0.0122	0.4253 ± 0.0143	2126 ± 71
F_4		0.7584 ± 0.0148	0.6976 ± 0.0125	0.4239 ± 0.0141	2119 ± 70
F_1	13	0.6132 ± 0.0282	0.4756 ± 0.0317	0.186 ± 0.0161	930 ± 80
F_2		0.6139 ± 0.0277	0.4785 ± 0.0307	0.1869 ± 0.0163	934 ± 81
F_3		0.6113 ± 0.0295	0.4764 ± 0.0326	0.1857 ± 0.0172	927 ± 86
F_4		0.6077 ± 0.0302	0.4736 ± 0.0348	0.1841 ± 0.0177	920 ± 88
F_1	22	0.4888 ± 0.0364	0.3495 ± 0.0367	0.1293 ± 0.0135	646 ± 67
F_2		0.4928 ± 0.0358	0.3547 ± 0.0353	0.1307 ± 0.0129	653 ± 64
F_3		0.4882 ± 0.0365	0.3509 ± 0.036	0.1290 ± 0.0133	644 ± 66
F_4		0.4866 ± 0.0349	0.3489 ± 0.0353	0.1286 ± 0.0127	642 ± 63
F_1	71	0.2660 ± 0.0337	0.1737 ± 0.0303	0.0579 ± 0.0097	289 ± 48
F_2		0.1609 ± 0.0339	0.1763 ± 0.03	0.0585 ± 0.0098	292 ± 49
F_3		0.2646 ± 0.0344	0.1741 ± 0.0291	0.0577 ± 0.0096	288 ± 48
F_4		0.2650 ± 0.0335	0.174 ± 0.0297	0.058 ± 0.0097	289 ± 47

The results in Tables 9.1 and 9.2 clearly show that the characteristics of the fitness landscape vary only slightly when the target (learning) function is changed from F_1 to F_4. In fact, the autocorrelation results suggest that the fitness landscape may become slightly smoother from F_1 to F_4, though the change is not statistically significant. It is not entirely surprising. On the interval of interest all four functions have similar values, as depicted in Figure 9.1. When sampling the values of the function to build the fitness cases, the fitness cases are likely to be similar. Thus if an individual approximates one function well, it is likely to approximate the other three as well. Thus it appears that the increasing difficulties of the problem instances stem directly from the increasing structural complexity of the target functions, rather than from a change in the ruggedness of the fitness landscape, supporting the arguments in chapter 5 regarding the scalable difficulty of the families of polynomial functions. However this discussion is not regarded as conclusive; future work, detailed at the end of the chapter, is intended to provide more evidence for that argument.

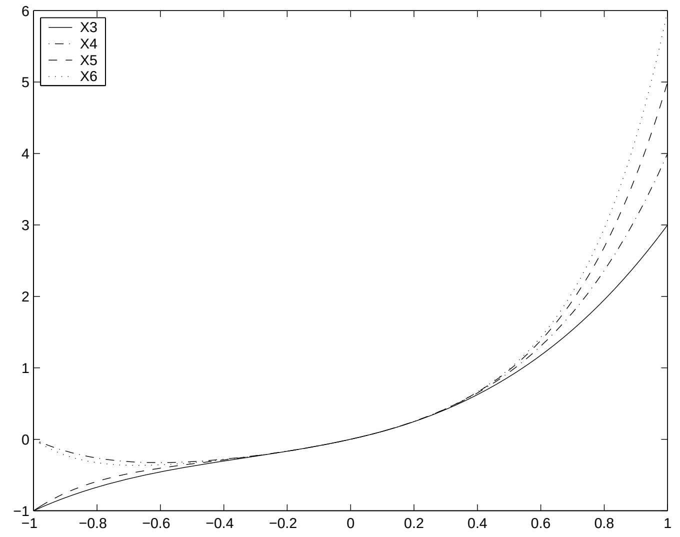


Figure 9.1: Graph of functions $F_1(X3)$ to $F_4(X6)$

For the second experiment, Table 9.3 and 9.4 show the autocorrelation values and information content for the random walks for the 6-Multiplexer problem.

Table 9.3. Autocorrelation analysis for 6-Multiplexer problem.

Length	G_1	G_2	G_3	G_4
1	0.8640 ± 0.0325	0.8899 ± 0.0242	0.901 ± 0.0193	0.9057 ± 0.0208
2	0.7719 ± 0.0450	0.8133 ± 0.0372	0.8291 ± 0.0301	0.8361 ± 0.0334
3	0.7131 ± 0.0531	0.7531 ± 0.0459	0.7715 ± 0.0387	0.7797 ± 0.0428
4	0.6603 ± 0.0581	0.7041 ± 0.0519	0.7236 ± 0.0048	0.7326 ± 0.0496
5	0.6165 ± 0.0618	0.6656 ± 0.0561	0.6826 ± 0.0497	0.6924 ± 0.0551
6	0.5796 ± 0.0644	0.6269 ± 0.0595	0.6473 ± 0.0537	0.6574 ± 0.0596
7	0.5475 ± 0.0665	0.5958 ± 0.0624	0.6165 ± 0.0570	0.6268 ± 0.0630
8	0.5193 ± 0.0677	0.5681 ± 0.0645	0.5888 ± 0.0598	0.5994 ± 0.0662
9	0.4945 ± 0.0686	0.5437 ± 0.0664	0.5642 ± 0.0621	0.575 ± 0.0688
10	0.4723 ± 0.0693	0.5216 ± 0.0681	0.5420 ± 0.0642	0.5533 ± 0.0712

Table 9.4. Information content analysis for 6-Multiplexer problem.

Function	ϵ	$H(\epsilon)$	$h(\epsilon)$	$M(\epsilon)$	Optima No.
G_1	0	0.6052 ± 0.05	0.4389 ± 0.0463	0.1832 ± 0.028	915 ± 140
G_2		0.5586 ± 0.0436	0.3898 ± 0.0404	0.1575 ± 0.0213	787 ± 106
G_3		0.5377 ± 0.0435	0.3708 ± 0.0416	0.1468 ± 0.0206	734 ± 103
G_4		0.5101 ± 0.045	0.3426 ± 0.0428	0.1342 ± 0.0195	670 ± 97
G_1	1	0.5179 ± 0.055	0.3451 ± 0.0504	0.1428 ± 0.0268	713 ± 134
G_2		0.4808 ± 0.0486	0.3099 ± 0.0436	0.1245 ± 0.0211	622 ± 105
G_3		0.4488 ± 0.0469	0.2804 ± 0.0408	0.110 ± 0.0186	554 ± 93
G_4		0.4268 ± 0.0483	0.2621 ± 0.0430	0.1023 ± 0.0183	511 ± 91
G_1	2	0.3652 ± 0.0507	0.1988 ± 0.0373	0.0849 ± 0.0156	424 ± 93
G_2		0.3445 ± 0.0473	0.1877 ± 0.0362	0.0763 ± 0.0162	381 ± 81
G_3		0.2959 ± 0.043	0.1513 ± 0.0307	0.0603 ± 0.0128	301 ± 64
G_4		0.2971 ± 0.0456	0.1535 ± 0.0340	0.0609 ± 0.0135	304 ± 67
G_1	8	0.0477 ± 0.0115	0.0136 ± 0.0039	0.0053 ± 0.0018	26 ± 8
G_2		0.0320 ± 0.0148	0.0086 ± 0.0147	0.0036 ± 0.0022	18 ± 11
G_3		0.0196 ± 0.01	0.0048 ± 0.0028	0.002± 0.0013	10 ± 6
G_4		0.0227 ± 0.0134	0.0058 ± 0.0041	0.0025 ± 0.0019	12 ± 9
G_1	13	0.017 ± 0.0072	0.004 ± 0.0019	0.0017± 0.009	8 ± 4
G_2		0.0167 ± 0.0127	0.0041 ± 0.0032	0.0017 ± 0.0014	8 ± 7
G_3		0.0089 ± 0.0074	0.002 ± 0.0018	0.0009 ± 0.0008	4 ± 4
G_4		0.0122 ± 0.0104	0.0029 ± 0.0028	0.0012 ± 0.0012	6 ± 6
G_1	18	0.003 ± 0.0002	0.0006 ± 0.0005	0.0003 ± 0.0002	1 ± 1
G_2		0.0006 ± 0.0001	0.0001 ± 0.0002	0 ± 0	0 ± 0
G_3		0.0002 ± 0.0001	0 ± 0.0001	0 ± 0	0 ± 0
G_4		0.0002 ± 0.0001	0 ± 0.00001	0 ± 0	0 ± 0

The results of both autocorrelation analysis and information content analysis consistently show that the ruggedness of the fitness landscape decreases as the grammar changes from G_1 to G_4, indicated by the increase in autocorrelation values and the decrease in information content (H) and partial information content (M). The statistical significance was tested using the one-tailed test for differences between two binomial variables, with confidence level $\alpha = 0.05$. For small autocorrelation length and small ϵ, when the analyses are most sensitive, the differences between G_1 and G_4 were found to be significant. Thus the bias induced by changing the grammar may have influenced the search not only by reducing in the search space size, but also by changing the characteristics of the fitness landscape. The improved performance of CFG-GP with the stronger biases was not solely due to the reduction in search space as claimed in [Whi1996], but also to the resulting smoothing of fitness landscape.

9.5 Conclusion

In this chapter, we argued that because of the fixed-arity tree structure of standard GP, and the even more constrained nature of GGGP derivation trees, it is difficult to define a topological structure, or to design operators that respect this structure. By transforming to the space of TAG-derivation trees, it is easier to define a natural topology and design operators that respect this topological structure, making small and bounded changes. Moreover, since the mapping from the genotype space to the phenotype space possesses the locality property, the changes on the phenotype space respect the corresponding topology in the phenotype space. Using insertion, deletion and point replacement, it is possible to investigate problem difficulty through characterising the fitness landscape on the genotype space, and in effect also on the original (phenotype) search space.

We showed some experiments which, for the first time in the literature, characterise the fitness landscape on a syntactically constrained domain. We were able to distinguish the effects of fitness landscape and of structural complexity on the problem difficulty. In addition, we investigated the effect on fitness landscape of changing search space bias by changing grammar, showing that the changing bias smoothes the fitness landscape in addition to reducing the search space. The results have shed some light on the performance of CFG-GP, as well as TAG3P, on those problems.

In the fitness landscape study for symbolic regression, future work includes a study of autocorrelation near the optima. With the fitness function used for these symbolic regression problems (total error over 20 sampling points), the search landscapes far from the optima are likely to be very similar for all functions, because most randomly sampled individuals are likely to be very far from any of the functions, so that the differences between the functions are masked. The similar autocorrelation values may stem simply from most sampled individuals being very far from the optima; but GP runs for these symbolic regression problems almost invariably converge to a small error within a few generations, so that most of the search is conducted close to the optima, and fitness landscapes far from the optima give a poor indication of the overall problem difficulty. There are some indications from the similar basin-density information values (h) in the information content analysis that the fitness landscapes may also be similarly close to the optima, but further work is needed to confirm this.

Chapter 10

An Alternative Comparison between Different Genetic Programming Systems

Since Koza's initial book on genetic programming (GP) [Koz92], a wide range of new genetic programming systems have been proposed. Typically, when each new system is introduced, it is compared with existing GP systems. The comparisons usually report on the new system's better performance over standard GP when solving particular problems. The reports contain descriptive statistics, such as cumulative frequencies, number of independent runs and the number of individuals that must be processed to yield a success with 99% probability. However, it is generally the case that the new system differs from previous systems over a number of dimensions (search space size, structure and representation, evolutionary operators, feasibility constraints, search algorithm, genotype-to-phenotype map, decoding, evaluation etc.). While reporting the above statistics is important, we agree with [Dai1997a, Dai1997b, Hay1998c] that it is also necessary to understand the causes of the differences. It is all too easy to assign the improvement from a new system to differences in representation or operators when simple changes in search space size may be more important. In particular, later in the chapter it will be shown how different types of bounds for chromosome complexity in bounded search spaces can be an important contributor to differences between GP systems, potentially masking the effects of the underlying representation changes.

In this chapter, it is argued that the multi-objective framework can help to solve some of these difficulties. As a test case, TAG3P is compared with standard GP on two standard problems from the literature, using a multi-objective selection mechanism. The chapter proceeds as follows. In the first section, the problem of different types of search complexity bounds is formulated. A brief introduction to multi-objective evolutionary optimization (EMO) is given. The third section contains a discussion of the use of multi-objective techniques to compare TAG3P and standard GP in search spaces of equivalent size. The experiments and discussion are presented in the fourth section. Finally, section 5 concludes the chapter and highlights some future work. Some of the material in this chapter has appeared in [NXH2004a].

10.1 Difficulties in Making Meaningful Comparison between Genetic Programming Systems

In this section, we discuss why it is generally difficult to make meaningful comparison between TAG3P and standard GP (or GGGP). Although the arguments presented here are based on TAG3P only, it is believed that they are equally applicable to a wide range of other genetic programming systems.

TAG3P, presented in chapter 4, differs in many ways from standard GP. It uses grammars (a LTAG and a CFG), can solve typed problem domains, can handle context-sensitive information, has a genotype-to-phenotype map (therefore different search space), has different genetic operators, and a different type of bound on chromosomal complexity (length rather than depth). If TAG3P and GP performance differ (as seen in chapter 5), it could be a result of any or all of these differences. Consequently, understanding the relationship between TAG3P and GP performance is very challenging.

Firstly, to ensure the grammars give no favorable bias for TAG3P they are chosen as follows. From a description of a GP set of functions and terminals, a context-free grammar, G, is created according to [Whi1996] (page 130) thus ensuring G is bias-free and the correspondence between derivation trees of G and parse tree of GP is one-to-one. G_{lex} is then derived from G using the algorithm in appendix A. Secondly, in order to evaluate fitness of an individual (derivation tree in G_{lex}), it is decoded first into a derivation tree in G, then to the equivalent parse tree in GP. On this final parse tree the evaluation is systematically processed in the same way as in GP. Next, tunable parameters in the two systems are set as uniformly as possible.

However, GP systems usually use a bounded search to limit chromosome complexity. In TAG3P, the bound is the maximum number of nodes, whereas in standard GP it is the maximum allowed depth (although some recent GP systems also use the maximum number of nodes [Lan2002]). It is virtually impossible to adjust these bounds to give the same phenotype space in each, because there is no systematic mapping between nodes in TAG3P elementary trees and nodes in GP. This problem is not restricted to TAG3P; we believe it applies equally to a range of other GP systems, especially those that use grammars and/or genotype-to-phenotype mappings (e.g Linear GP [Ban1998], GE, and GEP). Therefore, while the work in this chapter relates only to the problem of making comparison between TAG3P and GP, it has clear implications for other GP systems.

10.2 Multi-objective Evolutionary Optimization

Multi-objective evolutionary optimization (EMO) are algorithms that use evolutionary computation techniques to solve optimization problems with more than one objective to be optimized [Coe2002]. The purpose of EMO is not to obtain a single solution but to find a set of non-dominated solutions called "Pareto frontier". A solution a is called "Pareto-dominance" of a solution b if a is better than

b in all objectives. A solution a is called a (Pareto) non-dominated solution if it is not Pareto-dominated by any other solution. In other words, the task of EMO is to find a set of solutions that each solution in that set is not Pareto-dominated by any other solutions. A weak version of Pareto-dominance is called weak Pareto-dominance. A solution a is called weak Pareto-dominance of a solution b if a is better or equal than b in all objectives. The overview of EMO on problems, algorithms, techniques can be found in [Coe2002, Deb2002].

10.3 The Use of Multi-objective Techniques for Comparisons

To solve the problem of adjusting chromosome complexity bounds, one option might be to remove the bounds. However, unbounded GP systems usually bloat, which is why bounds on the chromosome complexity are usually set. Bloat is a well-documented phenomenon in GP [Bli1994, Ban1998, Sou1999, Lan2002]; It is suggested in [Lan2002] that code bloat is inevitable for all evolutionary systems that use length-variant chromosomes. But code bloat has serious effects on search performance [Bli1994, Sou1999, Lan2002]; and there is no reason to expect that these effects are invariant between different GP systems. Hence removing bounds simply adds one more confounding variable. Equally, when solving inductive learning problems at least, one is not indifferent to solution size - short solutions are preferred [Mit1997].

One alternative is to use a combined objective. Although there are many ways to combat bloat with single objective selection by integrating chromosome complexity into the fitness function [Iba1994, Zha1995, Bli1996], this introduces significant problems for our purpose. When using chromosome complexity as part of a single objective fitness function, some tunable parameters must be defined to determine how much the chromosome complexity of an individual will affect its fitness. It is at least difficult and time-consuming to find an optimal setting for these parameters; it is virtually impossible to find one which is fair to both systems. Hence, we argue that multi-objective selection is more appropriate for this purpose, especially since, in [Ble2001], it is shown that multi-objective selection can outperform single objective selection in combating bloat. To understand the effect of the difference in search space representation and operators between TAG3P and standard GP, we use an unbounded search space, but apply multi-objective selection, with chromosome complexity as the second objective, to combat bloat. This has two main advantages. Searching with this multi-objective selection pressure can solve the problem of code bloat [Bot2000, Lan1998, Ble2001, Eka2001, Dej2003], therefore permit a more meaningful comparison. Moreover, multi-objective selection can unify very different GP-search spaces into an uniform search problem, providing a common ground for looking into the search performance and behavior of GP systems that use different representations and different genetic operators.

In this chapter, we use the strength Pareto evolutionary algorithm (SPEA2) [Zit1999, Zit2001], a state-of-the-art evolutionary multi-objective optimisation (EMO) algorithm, to implement the comparison between GP and TAG3P. We chose SPEA2 because of its superior performance over other EMO algorithms [Zit1999, Zit2001] and its efficiencies in reducing bloat in GP [Ble2001]. Moreover, Pareto fitness calculation in SPEA2 uses density estimation techniques, helping to promote diversity in the objective space. In turn, that reduces the common effect where the whole population may converge to the individual with minimal chromosome complexity [Dej2003]. This was sufficiently effective that in our experiments using SPEA2, we did not observe the effect. An outline of SPEA2 is given in Appendix D of the thesis. It is also possible to use other EMO algorithms such as those in [Coe2002, Deb2002], and we intend to investigate alternatives in the future.

10.4 Experiments and Results

Using the SPEA2 multi-objective selection, we compared TAG3P with standard GP on two standard test problems: the 6-multiplexer, and symbolic regression problems. The descriptions of those standard problems were given in chapter 5. They were chosen as frequently-used GP test-beds, the (shortest) solutions being known. The fitness values of the first are discrete and bounded; of the second, continuous and unbounded.

10.4.1 Experiment Design

As in [Ble2001], we used weak Pareto non-domination selection. The second objective is the size (in number of nodes) of the GP expression trees, on which fitness is evaluated. To study the effect of population initialisation, for each problem, we ran three systems. The first was TAG3P, the second was GP with the initial population translated from the initial population of TAG3P (GP-I) such that they had the same initial population in the phenotype space, and the third was GP with Ramped-Half-and-Half initialisation (GP-RHH). To investigate the effect of variation in population size, and number of generations, for each problem, we experimented with three settings of population and number of generations. All other parameters for all three systems (such as genetic operator rates) are the same as in the single objective experiments in chapter 5. We can summarise the parameter settings in two categories:

Fixed common settings: Types and rates of genetic operators are the same as in chapter 5 for all systems. In SPEA2 settings, weak dominance was used and the size of the archive was set equal to the size of the main population. Very large maximal limits were set on the size and depth of individuals in TAG3P and GP respectively, to ensure, in effect, that there is no maximal limits on their size and depth. All runs only finished when the maximum number of generations was reached.

Varied common settings: On each problem, three varied settings of population size and number of generations were used with all three systems (TAG3P, GP-I and GP-RHH). The experimented population sizes were 250, 500, and 1000, while the maximum numbers of generations were 101, 51, and 26 respectively. There were 100 runs allocated for each system for each varied setting, making the total number of runs 1800. The ith run of each setting used the same initial random key as every other ith run. The keys themselves were generated at random. All the runs used the same random number generator.

10.4.2 Results

Table 10.1 and 10.2 show the results of all three systems with three different settings. The last three columns are the proportion of success, the average solution sizes (AvgSS), and the standard deviation of solution sizes (StdSS).

Table 10.1.Results on the symbolic regression problem.

Population	Systems	Success Rates	AvgSS	StdSS
250	TAG3P	45%	13.64	1.48
250	GP-I	16%	13.75	1.71
250	GP-RHH	5%	13.4	0.8
500	TAG3P	57%	13.68	1.67
500	GP-I	24%	14.17	1.82
500	GP-RHH	5%	15	3.1
1000	TAG3P	71%	13.62	1.35
1000	GP-I	35%	13.91	1.59
1000	GP-RHH	7%	14.86	3.8

Table 10.2. Results on the 6-multiplexer problems.

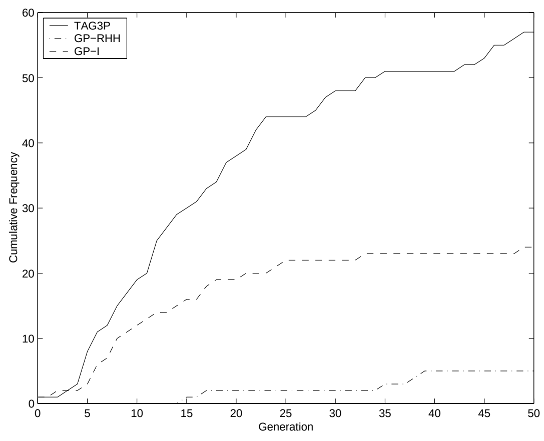
Population	Systems	Success Rates	AvgSS	StdSS
250	TAG3P	9%	21.67	7.06
250	GP-I	99%	14.49	6.47
250	GP-RHH	99%	14.28	7.9
500	TAG3P	30%	17.7	4.71
500	GP-I	97%	11.12	0.68
500	GP-RHH	95%	14.57	7.37
1000	TAG3P	30%	17.67	6.28
1000	GP-I	99%	11.20	0.71
1000	GP-RHH	87%	19.25	14.70

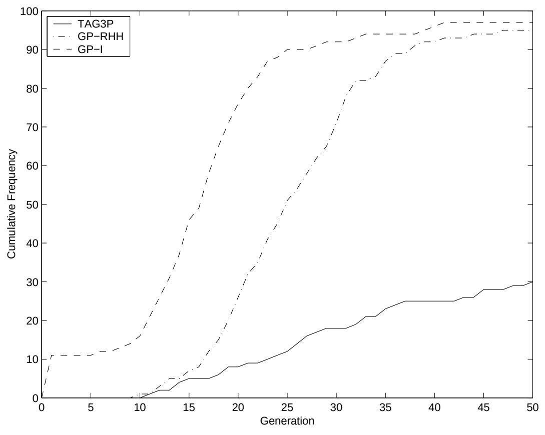
Graphical representations of the results for POPSIZE=500 are presented in figures 10.1, 10.2, and 10.3 depicting the cumulative frequencies (of how often each system found individuals with fitness is 0), average first fitness (the first fitness is the fitness measured against fitness cases of the problems, and average second fitness for each system (the second fitness of each individual is the size of the individual). The figures for POPSIZEs of 250 and of 1000 are sufficiently similar that they are given in Appendix E of the thesis.

10.4.3 Discussion of Results

The results given in this section show some remarkable differences from those in chapter 5, in which bounded search spaces and single objective selection pressure were used.

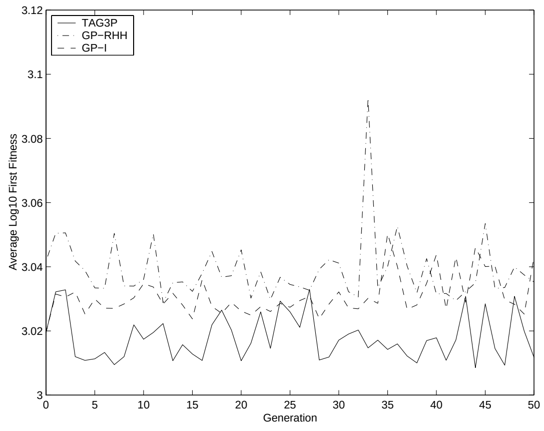
For the symbolic regression problem, the results in chapter 5 showed that TAG3P performance (93% success) was far better than for GP (9% success), for a population size of 500. As mentioned earlier, this difference in performance could come from two causes. The first possibility is the new representation and therefore, new operators. The second is the type and value of complexity bound on individual programs in the population. As previously noted, it is desirable to

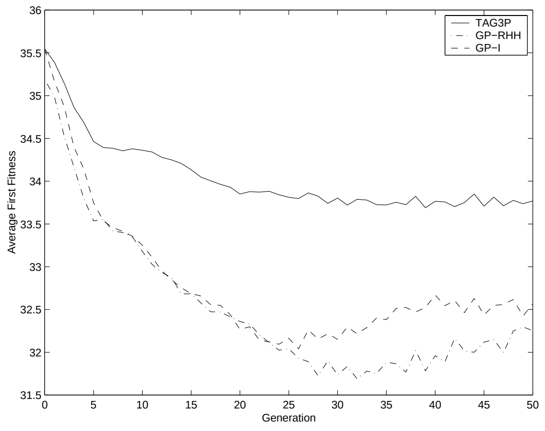




$SymbolicRegression$ $6 - Multiplexer$

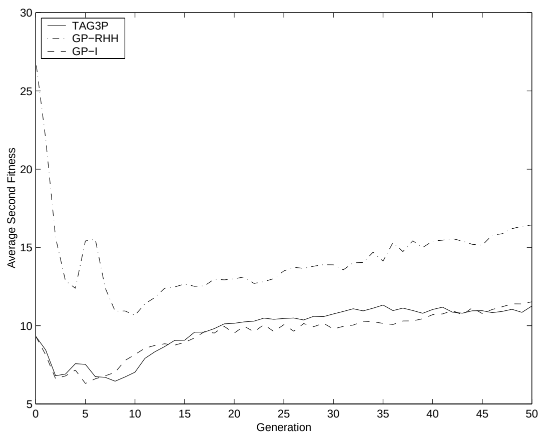
Figure 10.1: Cumulative Frequencies (POPSIZE=500)

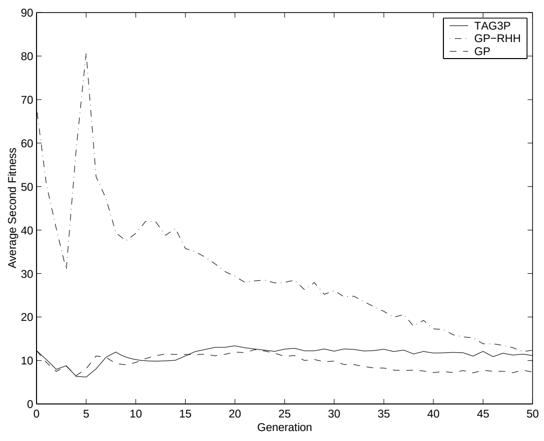




$SymbolicRegression$ $6 - Multiplexer$

Figure 10.2: Average First Fitness (POPSIZE=500)





$SymbolicRegression$ $6 - Multiplexer$

Figure 10.3: Average Second Fitness (POPSIZE=500)

separate out the effects of these two causes on the relative performances of the genetic programming systems. The difference in performance between TAG3P and GP-RHH in this section is 57% to 5%, a smaller gap compared to that in chapter 5. We emphasize that, for population size 500, all the parameter settings are the same for this experiment and that presented in chapter 5. This difference indicates that although TAG-representation and its genetic operators played an important role in helping TAG3P achieve a superior performance over GP, the difference in type and value of complexity bounds on individual programs also contributed. In other words, TAG3P performs much better than GP on symbolic regression problem, but if the difference in bounds setting is removed, the difference is significantly reduced.

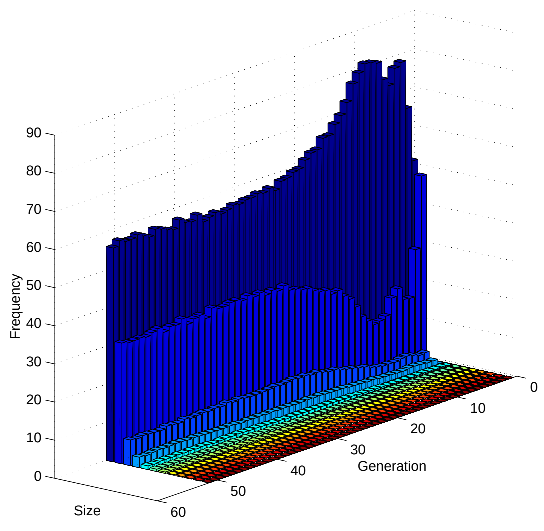
For the 6-multiplexer, the results show that the effect of differences in complexity bounds is much larger than the effect of different representations (operators). For population size 500, in chapter 5, the performances of TAG3P and GP are similar (63% and 63% success). However, the results in the case of unbounded complexity space and multi-objective selection pressure present a completely different story. The performance of GP improved to 99%, while it worsened to 30% for TAG3P. This implies that the use of different types and values of bounds in chapter 5 has played a vital role in promoting the performance of TAG3P while degrading the performance of GP.

The overall picture from comparing results in this chapter and those in chapter 5 is that TAG3P's new representation and operators worked better than GP's in reliably finding solutions, as well as in finding shorter solutions, on the symbolic regression problem, but that the reverse is true on the 6-multiplexer problem. However, in practice, the setting of bound values (and differences in types of bounds) on the complexity of individual programs can mask the effect of representation and operators.

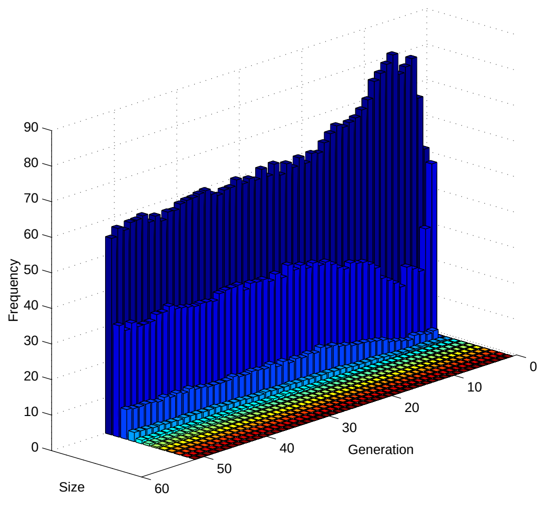
But how could the differences in types and values of complexity bounds influence the performance of a GP system? One possible explanation is that it changes the effects of code bloat. If the complexity bound is small, code bloat is held in check by the bound, while a large bound permits bloat. However the speed of bloat differs between different systems. For instance, a small bound is likely to cause the system to reach the bound quickly, and to bloat the code size of almost the whole population close to the bound. Using multi-objective selection pressure to combat bloat, and consequently removing the complexity bounds, can shed light on the differing biases toward code bloat of the different representations used by various GP systems.

Figures 10.4 and 10.5 show how the distribution of tree sizes in TAG3P and GP changed during their evolutionary process. The figures here are for the case POPSIZE=500 only. The figures for other cases are similar and given in Appendix E.

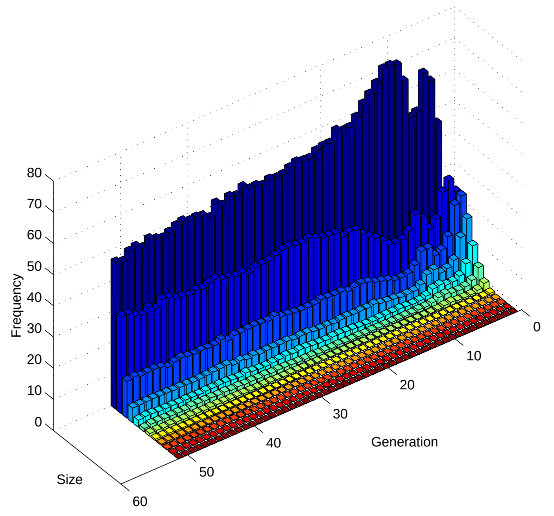
The figures show the three systems have different biases towards code bloat. TAG3P and GP-I have similar trends in evolving towards small trees. However the TAG3P population converged faster and more reliably than GP-I on the symbolic regression problem, while the converse is the case for the 6-multiplexer problem. The differences in the evolution of the tree size distribution between GP-I and GP-RHH highlight that the initial population is also an important factor for the bias in code bloat and speed of bloat in each system. When searching



$TAG3P$



$GP-I$



$GP-RHH$

Figure 10.4: Tree Size Frequencies for Symbolic Regression Problem, POPSIZE=500

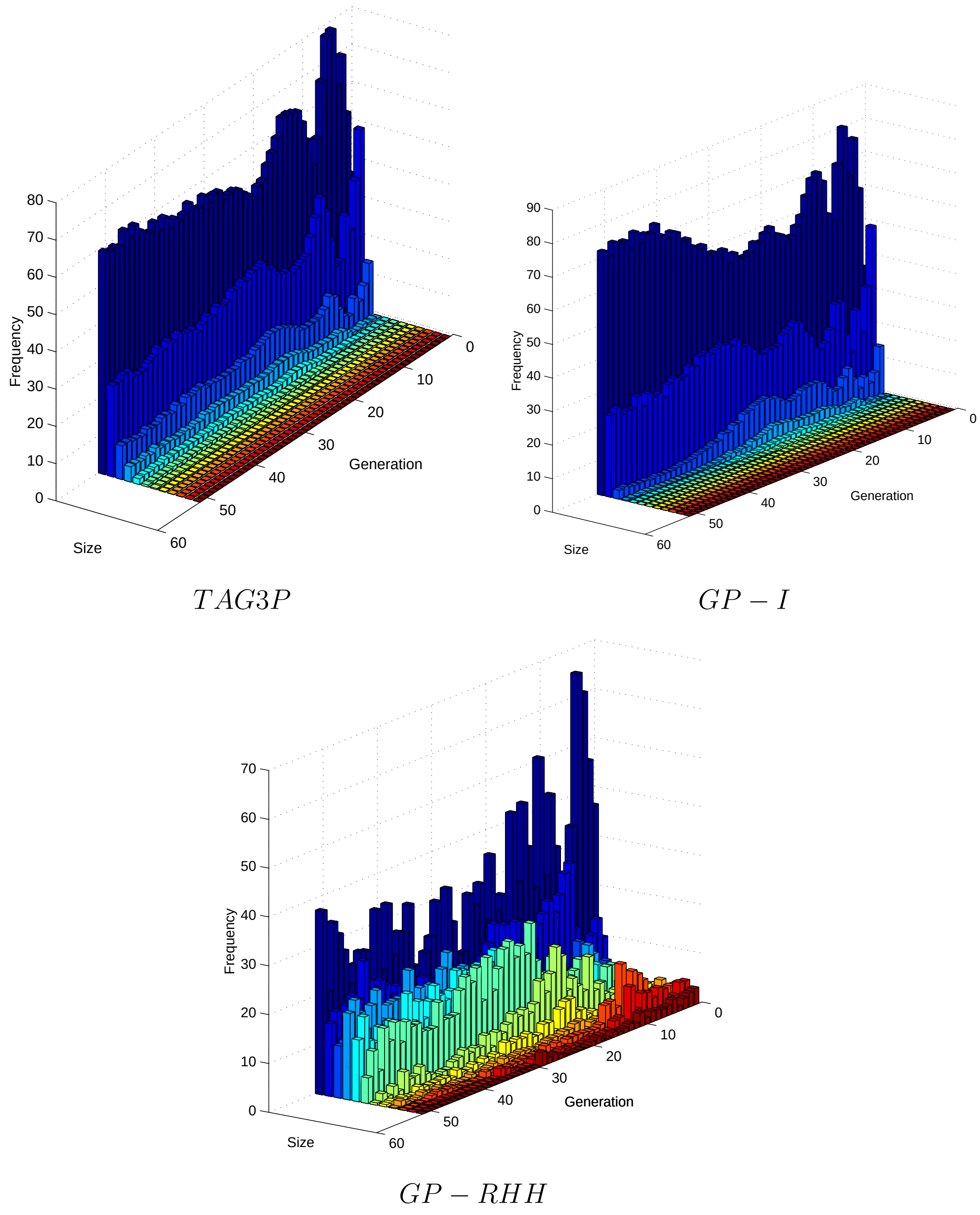


Figure 10.5: Tree Size Frequencies for 6-multiplexer Problem, POPSIZE=500

in spaces with unbounded complexity of individual programs, and under multi-objective selection pressure, it can make an impact on the performance of the systems. Referring to the results in table 10.1 and 10.2, the performance of GP-I and GP-RHH are statistically different in 15 out of 18 settings. This difference is interesting, as it contrasts with the findings in [Luk2001], where the seeding method used to generate the initial population had no significant impact on the search performance and behavior of GP, in a bounded search space and under single objective selection pressure.

10.5 Conclusion

In this chapter, we have addressed the issue of making an alternative type of comparison between different GP systems. In particular, we pointed out that different types of bounds on chromosome complexity, which might derive from different representations, make it hard to determine the causes of different performance between GP systems. We have argued that the use of multi-objective selection, coupled with an unbounded search space, can help to understand the effects of search space size. Thus, by using an EMO algorithm (SPEA2), we were able to make this alternative type of comparison between TAG3P and standard GP on two standard problems. The results show the differences in search performance and behavior between TAG3P and standard GP. We have also found that setting different types and values of bounds on search spaces was one factor in the better performance of TAG3P over GP in chapter 5, and on one problem (6-multiplexer) it masked the effect of representation and operators. Moreover, the method for seeding the initial population in GP can be important, contrary to other results. Although the results were based on a comparison between TAG3P and standard GP systems, we believe that similar problems and phenomena could appear in comparisons between other GP systems.

Future work includes the application of the alternative comparison technique in this chapter to other GP systems beyond TAG3P and standard GP. We also plan to apply other EMO algorithms, to confirm (as we anticipate) that the results and arguments of this chapter are not dependent on a particular algorithm.

Chapter 11

Conclusions and Future Work

This thesis has presented a number of issues stemming from the lack of operators in tree-based GP and GGGP that can make small and bounded changes. It has also noted that a number of linear representations for GP and GGGP in the literature do not respect the causality (or locality) principle of representation for EAs, so that although there are operators that can make small and bounded changes on the genotype space, the resulting changes on the phenotype space might not be bounded or controllable.

Motivated by a state-of-the-art formalism (TAG) from the field of natural language processing, the thesis has presented the design of a new representation for GP and GGGP, and investigated its properties. The new representation facilitates a range of new operators, both bio-inspired operators and minimal-change operators. Since the genotype-phenotype mapping is shown to be causal (local), the phenotype space effects of these minimal-change operators will also be small and bounded.

These properties, in turn, have been shown to simplify the characterisation of fitness landscapes on syntactically constrained domains, and to ameliorate the GP structural difficulty problem. The new TAG3P system based on this representation has been shown to be competitive with standard GP and GGGP systems. A schema theorem was formulated and proven, serving as a theoretical basis for TAG3P.

Finally, an alternative framework for making comparisons between different genetic programming representations, using different methods for bounding the search space, was proposed.

11.1 Contributions of this thesis

New Concept of TAG Derivation Tree: Although there have been a number of definitions of TAG-derivation trees, which have been shown to be useful in the context of linguistics and natural language processing, they are not directly suitable for the purpose of representing individuals in GP. In chapter 3, a new concept of TAG-derivation tree was proposed, unifying ideas from two well-known definitions.

New representation for GP and GGGP: A new and flexible representation for GP and GGGP was designed, based on the new version of TAG-derivation trees from chapter 3.

Locality Property: In chapter 3, the new TAG-based representation was shown to possess an important feature for genotype-phenotype mappings, namely locality. The locality (or causality) property allows small changes on the genotype space to result in small changes in the phenotype space, which this thesis has demonstrated to be useful in a number of situations.

The Importance of Making Small and Bounded Changes: The importance of the capacity to make small and bounded changes is addressed in this thesis. In particular, we argued that a number of difficulties in GP and GGGP – structural difficulty, difficulty of conducting fitness landscape analysis, lack of local search – arise from a lack of this capacity.

New GGGP System: TAG3P, a new grammar guided genetic programming system, was presented in chapter 4. Some experimental results on standard problems have indicated the robustness of the system.

New Operators: Nature employs a number of special genetic operators in the process of genome evolution. It is interesting and potentially useful to investigate how bio-inspired versions of these operators could be applied in GP. Chapter 6 has shown how some bio-inspired operators can be implemented in TAG3P, and investigated some of the contexts in which they are useful. There have been a number of previous implementations of similar operators in GP and GGGP systems, falling into two camps:

- problem-specific operators which cannot generalise to other problems
- GP systems employing a linear to tree mapping

We argue that the non-causal nature of the latter mappings means that the bio-inspired operators at the genotypic level may cause random and uncontrollable variation at the phenotypic level, throwing their biological provenance into question.

New Schema Theorem: Schema theories have been an important tool for gaining a theoretical understanding of evolutionary algorithms. The thesis has noted some limitations of the previous schema theory for GGGP, and proposed a new concept of schema on TAG-based representation that unifies three important aspects of schemata for syntactically constrained domains. A simple schema theorem was stated and proven in chapter 7, serving as a theoretical basis for the TAG3P system.

New Hypothesis on The Problem of Structural Difficulty in GP: Elaborating on previous work in the literature, a new hypothesis was proposed, in chapter 8 of the thesis, to explain why structure alone could be a source of problem difficulty in GP. In particular, we conjectured that the problem lies in the lack of structural minimal-change operators in standard GP, pointing to the fixed arity of its representation as the culprit. Using the new representation and new structural mutation operators, we gave some experimental evidence for the hypothesis.

Fitness Landscape Study on Syntactically Constrained Domains: In chapter 9, it was shown how operators that respect the topological structure on both genotype and phenotype space facilitate fitness landscape analysis on syntactically constrained domains. A fitness landscape analysis, to the best of our knowledge the first in GGGP, was conducted, resulting in one case in new insights into the way change of bias (grammar) affects problem difficulty, and in another, confirming previous understanding of the changing problem complexity in a standard family of symbolic regression test problems.

Alternative Comparison between Different Genetic Programming Systems: This thesis proposed, in chapter 10, a framework for using multi-objective selection pressure for comparison between different GP systems. The purpose of the comparison is to separate out two effects on performance: the effects of different search space sizes, and the effects of different representations and operators. In particular, we note that some performance differences between TAG3P and other GP systems, which we might well have attributed to the operators, were primarily the result of differences in search space sizes. We conjecture that a range of other large performance differences reported in the literature, especially those involving linear genotypes, might benefit from such analysis.

Connecting the field of natural language processing to GP: This thesis applies some important ideas from natural language processing to GP, in the process solving some important problems. It demonstrates the importance of the interplay between two seemingly distant fields of computer sciences.

11.2 Future Work

In this section, we highlight some directions for future work extending this thesis. Since the work in the thesis can be divided into two groups, on TAG-based representation and on TAG3P, the future directions are also grouped accordingly.

11.2.1 Future Work on TAG-based Representation

Investigation of the handling of long-distance dependencies in GP: The epistasis phenomenon, where the content of some genes are dependent on others in the chromosome, has been identified as a key source of problem difficulty in GAs. The higher degree of epistasis a problem has, the more difficult it is. In GP, the dependencies between primitives in a tree-based chromosome are an extension of epistasis in GAs [ORe1998]. However, as pointed out in [ORe1998], the dependencies in GP are harder to capture, since they can move far away from each other in an unpredictable fashion. However the purpose of the ELD and FRD properties of TAGs, described in chapter 3, is to capture long-distance dependencies in sentence structure. It is interesting and useful to see if ELD and FRD could helps TAG-based representations to solve problems requiring long distance dependencies in GP. Some preliminary results (not presented in this thesis) on an artificial problem, called ORDERTREE, which models long distance dependencies, have given some hints of a positive answer. However the positive results could also be due to some of the other effects of TAG operators discussed in this thesis. Further work is required to confirm whether the effects are in fact due to the FRD and ELD properties, as we currently surmise.

More fine-grained definition of schemata: The definition of schemata given in chapter 7 could be refined to include “don’t care” symbols in the nodes of tree-based schemata. With this refinement, it is hoped that tighter lower bounds on the schema theorem for TAG3P could be derived.

Study of other properties of Genotype-Phenotype mapping: This thesis has concentrated on one of the most important properties of genotype-phenotype mappings, namely the locality property. However, it is not the only important property of the mapping from genotypes to phenotypes. In [Shi1999, Sha2000], two other important properties of redundant genotype-phenotype mappings were identified, namely the connectedness and extensiveness of the induced neutral sets. It would be useful to examine these properties in detail for TAG-based representations.

Other Search Strategies: The search strategies used in this thesis were genetic search and greedy hill-climbing. Other work by the author [NXH2001c, Abb2002], has given preliminary results on ant colony optimization, as an alternative to genetic operators, in searching the space of GP problems. This work used a restricted linear form of TAG derivation trees, but extension to tree-based TAG representation is feasible. The positive early results indicate strong potential for further work in this direction.

11.2.2 Future Work on TAG3P

Using Subcode Fitness to Bias Operator Sites: In TAG3P, as shown in chapter 4 and 5 of the thesis, it is possible to measure quantitatively how each subcode contributes to the overall fitness of the individual containing it. This property could potentially be used to bias the selection of sites for application of genetic operators, allowing subcodes to compete for being affected by the operators. In turn, this could be used to make the genetic operators more efficient, by protecting useful subcodes and renovating subcodes which are harmful to the fitness of the individual.

Adaptive Use of Genetic Operators: In this thesis, we have presented a quite large variety of genetic operators. However their usefulness is problem-dependent, as their effects may be positive on one problem and negative on another. Chapter 6 gave some general recommendations on use of the operators, but they must be determined by the system users rather than by the system itself. A potentially important direction of research is to build adaptive and dynamic mechanisms to choose genetic operators based on their search performance. Useful guidance is likely to come from outside GP, in areas such as evolutionary strategies and genetic algorithms where adaptive operators have been widely studied.

Learning the Grammars. Work in [Whi1996] (chapter 6) and [O'Ne2004a] have highlighted the value of learning and adapting the language bias (grammar) during the evolutionary process. More recently, a novel grammar-model based learning system was proposed, incorporating the learning of grammars from the individuals in the population for program induction. Its promising grammar learning method was based on techniques from the field of grammatical inference and statistical natural language processing [Man1999]. At about the same time, similar techniques for learning a LTAG from sample trees were proposed in [Sch1992]. A potential direction of research is the incorporation of the algorithm given in [Sch1992] to adapt the grammars during the evolutionary process of TAG3P.

Real World Application: TAG3P has so far been applied to standard test problems in the GP field. Future work will aim to apply TAG3P in solving real world problems where its representation and operators appear to be suitable. One example lies in the field of bioinformatics, for example predicting and designing secondary RNA structures [Wat1995]. An indication of the potential was presented in [Kob1994, Yok1995], where some special TAGs were shown to be very useful in modeling RNA structures.

Bibliography

[Aar1989] E. Aarts and J. Krost, *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*, John Wiley & Sons, 1989.

[Aar1997] E. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*, John Wiley & Sons, 1997.

[Abb2002] H.A. Abbass et al, AntTAG: A New Method to Compose Computer Programs Using Colonies of Ants, *Proceedings of the Congress on Evolutionary Computation (CEC'2002)*, IEEE Press, 1654-1659, 2002.

[Abr1984] H. Abramson, Definite Clause Translation Grammars, in *Proceedings of IEEE Logic Programming Symposium*, IEEE Press, 233-240, 1984.

[Abr1989] H. Abramson and V. Dahl, *Logic Grammars*, Springer-Verlag, 1989.

[Abr1963] N. Abramson, *Information Theory and Coding*, McGraw-Hill, 1963.

[Aho1986] A.V. Aho et al, *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, MA, 1986.

[Alb1999] E. Alba et al, Evolutionary Design of Fuzzy Logic Controllers Using Strongly-Typed GP, *Mathware & Soft Computing*, **6**(1), 109-124, 1999.

[Alb2000] P. Albuquerque et al, On the Impact of the Representation on Fitness Landscapes in Genetic Programming, *Proceedings of The Fourth European Conference on Genetic Programming (EuroGP 2000)*, LNCS 1802, Springer-Verlag, 1-15, 2000.

[Ale2001] R. Aler et al, Grammars for Learning Control Knowledge with GP, *Proceedings of the Congress on Evolutionary Computation (CEC 2001)*, IEEE Press, 1220-1227, 2001.

[Alt1994] L. Altenberg, The Evolution of Evolvability in Genetic Programming, in K.E. Kinnear Jr, editor, *Advances in Genetic Programming*, MIT Press, chapter 3, 47-74, 1994.

[Alt1995] L. Altenberg, The Schemata Theorem and Price's Theorem, in L. Darrell Whitley and M.D. Vose, editors, *Foundations of Genetic Algorithms 3*, Morgan Kaufmann, 23-49, 1995.

[Ang1996] P.J. Angeline and K.E. Kinnear Jr, *Advances in Genetic Programming II*, The MIT Press, 1996.

[Ant1991] H.J. Antonisse, A Grammar-Based Genetic Algorithm, in G.J.E. Rawlins, editor, *Foundations of Genetic Algorithms*, Morgan Kaufmann, 1991.

[Aza2003] R.M.A. Azad and C. Ryan, Structural Emergence with Order Independent Representations, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003)*, LNCS 2724, 1626-1638, Springer-Verlag, 2003.

[Bac1996] T. Back, *Evolutionary Algorithms in Theory and Practice: Evolutionary Strategies, Evolutionary Programming, and Genetic Algorithms*, Oxford University Press, 1996.

[Bac1997] T.Back et al, *Handbook of Evolutionary Computation*, editors, IOP Publishing Ltd and Oxford University Press, 1997.

[Bac2000a] T.Back et al, editors, *Evolutionary Computation 1: Basic Algorithms and Operators*, IOP Publishing Ltd, 2000.

[Bac2000b] T.Back et al, editors, *Evolutionary Computation 2: Advanced Algorithms and Operators*, IOP Publishing Ltd, 2000.

[Ban1998] W. Banzhaf et al, *Genetic Programming: An Introduction*, Morgan Kaufmann, CA, 1998.

[Bar1998] G. Barnbrook, *Language and Computer: A Practical Introduction to the Computer Analysis of Language*, Edinburgh University Press, 1998.

[Ble2001] S. Bleuler et al, Multi-objective Genetic Programming: Reducing Bloat Using SPEA2, *Proceedings of the Congress on Evolutionary Computation (CEC 2001)*, 536-543, 2001.

[Bli1994] T. Blickle and L. Thiele L., Genetic Programming and Redundancy, in J. Hopf, editor, *Genetic Algorithms within the Framework of Evolutionary Computation*, 33-38, 1994.

[Bli1996] T. Blickle, Evolving Compact Solutions in Genetic Programming: A Case Study, in H.M. Voigt et al, editors, *Paralell Problem Solving fron Nature (PPSN IV)*, Springer-Verlag, 564-573, 1996.

[Boh1997] W. Bohm and A.G. Schultz, Exact Uniform Initialization for Genetic Programming, in R.K. Belew and M.D. Vose, editors, *Foundations of Genetic Algorithms 4*, Morgan Kaufmann, 379-408, 1997.

[Bot2000] M.C.J. Bot, Improving Induction of Linear Classification Tree with Genetic Programming, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2000)*, Morgan Kaufman, 403-410, 2000.

[Bra2002] A. Brabazon et al, Grammatical Evolution And Corporate Failure Prediction, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002)*, Morgan Kaufmann, 1011-1018, 2002.

[Bru1996] W. S. Bruce, Automatic Generation of Object-Oriented Programs Using Genetic Programming, *Genetic Programming 1996: Proceedings of the First Annual Conference*, MIT Press, 267-272, 1996.

[Bru2002] P. Bruhn and A.G. Schulz, Genetic Programming over Context-Free Languages with Linear Constraints for the Knapsack Problem: First Results, *Evolutionary Computation*, **10**(1), 51-74, 2002.

[Cha2000] O.A. Cha et al, Characterizing a Tunably Difficult Problem in Genetic Programming, *Proceedings of Genetic Algorithms and Evolutionary Computation Conference (GECCO 2000)*, Morgan Kaufmann, 395-402, 2000.

[Cho1956] N. Chomsky, Three Models for the Description of Language, *IEEE Transactions on Information Theory*, **2**(3), 113-124, 1956.

[Cle2002] M. Clergue et al, Fitness Distance Correlation and Problem Difficulty for Genetic Programming, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 02)*, Morgan Kaufmann, 724-732, 2002.

[Coe2002] C.A. Coello Coello et al, *Evolutionary Algorithms for Solving Multi-Objective Problems*, Kluwer Academic, 2002.

[Cov1991] T.M. Cover and J.A. Thomas, *Elements of Information Theory*, John Wiley & Sons, 1991.

[Cra1985] N.L. Cramer, A Representation for the Adaptive Generation of Sequential Programs, *Proceedings of an International Conference on Genetic Algorithms and the Applications*, 183-187, 1985.

[Dai1997a] J.M. Daida et al, Challenges with Verification, Repeatability, and Meaningful Comparison in Genetic Programming: Gibsons Magic, Accessed at http://citeseer.nj.nec.com/257412.html, Date: 11 Oct 2003.

[Dai1997b] J.M. Daida et al, Challenges with Verification, Repeatability, and Meaningful Comparisons in Genetic Programming, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, Morgan Kaufmann, 64-69, 1997.

[Dai1999] J.M. Daida et al, Analysis of Single-Node (Building) Blocks in Genetic Programming, in L. Spector, W.B. Langdon et al, editors, *Advances in Genetic Programming III*, The MIT Press, 217-241, 1999.

[Dai2001] J.M. Daida et al, What Makes a Problem GP-Hard? Analysis of a Tunably Difficult Problem in Genetic Programming, *Genetic Programming and Evolvable Machines*, **2**, 165-191, 2001.

[Dai2002] J.M. Daida, Limit to Expression in Genetic Programming: Lattice-Aggregate Modeling, *Proceedings of the Congress on Evolutionary Computation (CEC 2002)*, IEEE Press, 273-278, 2002.

[Dai2003a] J.M. Daida and A.M. Hilss, Identifying Structural Mechanism in Standard GP, *Proceedings of Genetic Algorithms and Evolutionary Computation Conference (GECCO'2003)*, LNCS 2724, Springer-Verlag, 1639-1651, 2003.

[Dai2003b] J.M. Daida et al, What Makes a Problem GP-Hard? Validating a Hypothesis of Structural Causes, *Proceedings of Genetic Algorithms and Evolutionary Computation Conference (GECCO 2003)*, LNCS 2724, Springer-Verlag, 1665-1677, 2003.

[Deb1997] K. Deb et al, Fitness Landscapes, in T.Back at el, editors, *Handbook of Evolutionary Computation*, **B2.7**, IOP Publishing Ltd and Oxford University Press, 1997.

[Deb2002] K. Deb, *Multi-objective Optimization using Evolutionary Algorithms*, John Wiley & Sons, 2002.

[Dej2003] E.D. DeJong and J.B. Pollack, Multi-Objective Methods for Tree Size Control, *Genetic Programming and Evolvable Machines*, 4, 211-233, 2003.

[Der1988] P. Deransart et al, *Attribute Grammars: Definitions, Systems, and Bibliography*, LNCS 461, Springer-Verlag, 1988.

[Dic1987] D. Dickmanns et al, Dergenetische Algorithmus: Eine Implementarung in Prolog, Fortgeschrittenenpraktikum, Institut, f.Informatik, Lehrstihl Prof. Radig, Tech.Univ.Muchich, 1987.

[Drl1984] K. Drlica, *Understanding DNA and Gene Cloning: A Guide for the Curious*, John Wiley & Sons, USA, 1984.

[Dro1998] S. Droste and D. Wiesmann, On Representation and Genetic Operators in Evolutionary Algorithms, Accessed at: citeseer.ist.psu.edu/droste98representation.html on 30 oct 2004.

[Eka2001] A. Ekart and S.Z. Nemeth, Selection Based on the Pareto Nondomination Criterion for Controlling Code Growth in Genetic Programming, *Genetic Programming and Evolvable Machines*, **2**(1), 61-73, 2001.

[Eka2002] A. Ekart and S.Z. Nemeth, Maintaining the Diversity of Genetic Programs, *Proceedings of the 5th European Conference on Genetic Programming (EuroGP 2002)*, LNCS 2278, Springer-Verlag, 162-171, 2002.

[Eka2004] A.E Ekart and S. Gustafson, A Data Structure for Improved GP Analysis via Efficient Computation and Visualisation of Population Measures, *Proceedings of the 7th European Conference on Genetic Programming (EuroGP 2004)*, LNCS 3003, Spinger-Verlag, 35-46, 2004.

[Fer2001] C. Ferreira, Gene Expression Programming: A New Adaptive Algorithm for Solving Problems, *Complex Systems*, **3**(2), 87-129, 2001.

[Fer2002a] C. Ferreira, Mutation, Transposition, and Recombination: An Analysis of the Evolutionary Dynamics, *Proceedings of The 4th International Workshop on Frontiers in Evolutionary Algorithms*, 614-617, 2002.

[Fer2002b] C. Ferreira, Analyzing the Founder Effect in Simulated Evolutionary Processes Using Gene Expression Programming, *Soft Computing Systems, Design, Management and Applications*, IOS Press, 153-162, 2002.

[Fer2002c] C. Ferreira, Genetic Representation and Genetic Neutrality in Gene Expression Programming, *Advances in Complex Systems*, **5**(4), 389-408, 2002.

[Fer2002d] C. Ferreira, Function Finding and the Creation of Numerical Constants in Gene Expression Programming, *Proceedings of the 7th Online World Conference on Soft Computing in Industrial Applications*, 2002.

[Fog1995] D.B. Fogel, *Evolutionary Computation: Toward a New Phylosophy of Machine Intelligence*,Wiley & Sons, New York, 1995.

[Fog1966] L.J. Fogel et al, *Artificial Intelligence through Simulated Evolution*, Wiley & Sons, New York, 1966.

[Fre1998] J.J. Freeman, A Linear Representation for GP using Context Free Grammars, *Proceedings of Genetic Programming 1998, the Third Annual Conference on Genetic Programming*, Morgan Kaufmann, 72-77, 1998.

[Fri1958] R. FriedBerg, A Learning Machine - part I, *IBM Journal of Research and Development*, **2**, 2-12, 1958.

[Fri1959] R. FriedBerg, B. Dunham, and J. North, A Learning Machine - part II, *IBM Journal of Research and Development*, **3**, 282-287, 1959.

[Fre2002] A. A. Freitas, *Data Mining and Knowledge Discovery with Evolutionary Algorithms*, Springer-Verlag, Germany, 2002.

[Gib1985] A. Gibbon, *Algorithmic Graph Theory*, Cambridge University Press, 1985.

[Glo1997] F. Glover and M. Languna, *Tabu Search*, Kluwer Academic Publishers, Boston, 1997.

[Gol1989] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, MA, 1989.

[Gol1998] D.E. Goldberg and U.M. O'Reilly, Where Does the Good Stuff Go and Why?, *Proceedings of The First European Conference on Genetic Programming (EuroGP)*, Springer-Verlag, 1998.

[Got1999] J. Gottlieb and G. R. Raidl, Characterizing Locality in Decoder-Based EAs for the Multidimensional Knapsack Problem, *Proceedings of Artificial Evolution*, LNCS 1829, Springer-Verlag, 38-52, 1999.

[Gri1986] R. Grishman, *Computational Linguistics: An Introduction*, Cambridge University Press, 1986.

[Gru1996] F. Gruau, On Syntactical Constraints with Genetic Programming, in P. Angeline, and K.E. Kinear Jr, editors, *Advances in Genetic Programming 2*, MIT Press, 402-417, 1996.

[Har1997] C. Harris, Enforcing Hierarchy on Solutions with Strongly Typed Genetic Programming, *Late Breaking Papers at the 1997 Genetic Programming Conference*, 292-297, 1997.

[Hay1995a] T. Haynes, Clique Detection via Genetic Programming, Technical Report UTULSA-MCS-95-02, University of Tulsa, 1995.

[Hay1995b] T. Haynes et al, Strongly typed genetic programming in evolving cooperation strategies, *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA95)*, Morgan Kaufmann, 271-278, 1995.

[Hay1996a] T. Haynes, Duplication of Coding Segments in Genetic Programming, Technical Report UTULSA-MCS-96-03, University of Tulsa, 1996.

[Hay1996b] T. Haynes et al, Type Inheritance in Strongly Typed Genetic Programming, in P.J. Angeline and K.E. Kinnear Jr, editors, *Advances in Genetic Programming 2*, MIT Press, 359-376, 1996.

[Hay1998a] T. Haynes, Collective Adaptation: The Exchange of Coding Segments,*Evolutionary Computation*, **6**(4), 311-338, 1998.

[Hay1998b] T.Haynes, *Collective Adaptation: The Sharing of Building Blocks*, PhD Thesis, Department of Mathematical and Computer Sciences, University of Tulsa, 1998.

[Hay1998c] T. Haynes, Perturbing the Representation, Decoding, and Evaluation of Chromosomes, *Genetic Programming 98: Proceedings of the Third Annual Conference*, Morgan Kaufmann, 122-127, 1998.

[Hem1994] H. Hemmi et al, Development and Evolution of Hardware Behaviours, *Proceedings of Artificial Life IV*, MIT Press, 371-376, 1994.

[Hao2004] Hoang Tuan Hao et al, Does it Matter Where you Start? A Comparison of Two Initialisation Strategies for Grammar Guided Genetic Programming, *Proceedings of The Second Asian-Pacific Workshop on Genetic Programming*, 2004.

[Hao2005] Hoang Tuan Hao et al, The Importance of Local Search, A Grammar Based Approach to Environmental Time Series Modelling, to appear in *Genetic Programmin: Theory and Practice III*, Springer-Verlag, 2005.

[Hol1975] J.H. Holland, *Adaptation in Natural and Artificial Intelligence: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, Michigan University Press, 1975.

[Hop1979] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata: Theory, Languages, and Computation*, Addison-Wesley, 1979.

[Hro2003] J. Hromkovic, *Algorithmics for Hard Problems*, Springer-Verlag, Germany, 2003.

[Hus1998] T.S. Hussain and R.A. Browse, Attribute Grammars for Genetic Representations of Neural Networks and Syntactic Constraints on Genetic Programming, *Proceedings of AIVNGI'98: Workshop on Evolutionary Computation*, 1998.

[Hus1999] T.S. Hussain and R. A. Browse, Genetic Operators with Dynamic Biases that Operate on Attribute Grammar Representations of Neural Networks, *Proceedings of Workshop on Advanced Grammar Techniques Within Genetic Programming and Evolutionary Computation*, 83-86, 1999.

[Iba1994] H. Iba et al, Genetic Programming Using a Minimum Description Length Principle, in K.E. Kninnear Jr, editor, *Advances in Genetic Programming*, MIT Press, 265-284, 1994.

[Ige1998] C. Igel, Causality of Hierarchical Variable Length Representations, *Proceedings of the 1998 IEEE World Congress on Computational Intelligence*, IEEE Press, 324-329, 1998.

[Jon1995a] T. Jones, *Evolutionary Algorithms, Fitness Landscapes, and Search*, PhD Thesis, University of New Mexico and Santa Fe Institute, 1995.

[Jon1995b] T. Jones, *One Operator, One Fitness Landscape*, Working Papers 95-02-025, Santa Fe Institute, 1995.

[Jon1995c] T. Jones and S. Forrest, Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms, *Proceedings of the 6th International Conference on Genetic Algorithms*, Morgan Kaufmann, 184-192, 1995.

[Jos1975] A.K. Joshi et al, Tree Adjunct Grammars, *Journal of Computer and System Sciences*, **10** (1), 136-163, 1975.

[Jos1977] A.K. Joshi, Constraints on Structural Descriptions: Local Transformation, *SIAM Journal of Computing*, June, 1977.

[Jos1985] A.K. Joshi, How Much Context-sensitivity is Necessary for Characterizing Structural Description, in D. Dowty et al, editors, *Natural Language Processing - Theoretical, Computational and Psychological Perspectives*, Cambridge University Press, 1985.

[Jos1987] A.K. Joshi, An Introduction to Tree Adjoining Grammars, in A. Manaster-Ramer, editor, *Mathematics of Language*, John Benjamins, Amsterdam, 1987.

[Jos1991] A.K. Joshi et al, The Convergence of Mildly Context-Sensitive Grammar Formalisms, in P. Sells et al, editors, *Foundation Issues in Natural Language Processing*, MIT Press, MA, 1991.

[Jos1997] A.K. Joshi and Y. Schabes, Tree Adjoining Grammars, in G. Rozenberg and A. Saloma, editors, *Handbook of Formal Languages*, Springer-Verlag, 69-123, 1997.

[Kar1995] H. Kargupta, Signal-to-noise, Crosstalks, and Long Range Problem Difficulty in Genetic Algorithms, *Proceedings of the 6th International Conference on Genetic Algorithms*, Morgan Kaufmann, 193-200, 1995.

[Kau1993] S.A. Kauffman, *The Origins of Order: Self-Organization and Selection in Evolution*, Oxford University Press, 1993.

[Kei1999] M. Keijzer and V. Babovic, Dimensionally Aware Genetic Programming, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999)*, Morgan Kaufmann, 1069-1076, 1999.

[Kei2001] M. Keijzer et al, Ripple Crossover in Genetic Programming, *Proceedings of the 4th European Conference on Genetic Programming (EuroGP 2001)*, LNCS 2038, Springer-Verlag, 74-86, 2001.

[Kei2002] M. Keijzer, M. O'Neill, C. Ryan, and M. Cattolico, Grammatical Evolution Rules: The Mod and the Bucket Rule, *Proceedings of the 5th European Conference on Genetic Programming (EuroGP'2002)*, LNCS 2278, Springer-Verlag, 123-130, 2002.

[Kei1994] M.J. Keith and C. Martin, Genetic Programming in C++: Implementation Issues, in K.E. Kinnear Jr, editor, *Advances in Genetic Programming*, MIT Press, 285-310, 1994.

[Kel1996] R. Keller and W. Banzhaf, GP Using Mutation, Reproduction and Genotype-Phenotype Mapping from Linear Binary Genomes into Linear LALR Phenotypes, *Genetic Programming 96*, MIT Press, 116-122, 1996.

[Kin1994a] K.E. Kinnear Jr, editor, *Advances in Genetic Programming I*, MIT Press, 1994.

[Kin1994b] K.E. Kinnear Jr, Fitness Landscapes and Difficulty in Genetic Programming, *Proceedings of the 1994 IEEE World Conference on Computational Intelligence*, IEEE Press, 142-147, 1994.

[Knu1968] D.E. Knuth, Semantics of Context-Free Languages, *Mathematical Systems Theory*, **2**(2), 127-145, 1968.

[Kob1994] S. Kobayashi and T. Yokomori, Modelling Secondary RNA Structures using Tree Grammars, *Proceedings of Genome Informatics Workshop V*, Universal Academy Press, 29-38, 1994.

[Koz92] J.R. Koza, *Genetic Programming: On the Programming of Computers by Natural Selection*, MIT Press, MA, 1992.

[Koz94] J.R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs*, MIT Press, MA, 1994.

[Koz99] J.R. Koza et al, *Genetic Programming III: Darwinian Invention and Problem Solving*, Morgan Kaufmann, CA, 1999.

[Koz2004] J.R. Koza et al, *Genetic Programming IV*, Kluwer Academic, 2004.

[Koz1995a] J.R. Koza, Evolving the Architecture of a Multi-part Program in Genetic Programming Using Architecture-Altering Operations, *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*, MA, MIT Press, 695-717, 1995.

[Koz1995b] J.R. Koza and D. Andre, Evolution of both the Architecture and the Sequence of Work-Performing Steps of a Computer Program Using

Genetic Programming with Architecture-Altering Operations, *Proceedings of American Association (AAAI)-95*, AAAI Press, CA, 1995.

[Koz1995c] J. Koza, Gene Duplication to Enable Genetic Programming to Concurrently Evolve Both the Architecture and Work-Performing Steps of a Computer Program, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 734-740, 1995.

[Koz1996] J. Koza and D. Andre, Classifying Protein Segments as Transmembrane Domains Using Architecture-Altering Operations in Genetic Programming, in P.J. Angeline and K.E. Kinear Jr, *Advances in Genetic Programming 2*, Chapter 8, MIT Press, 1996.

[Kro1985] A. Kroch and A.K. Joshi, Linguistic Relevance of Tree Adjoining Grammars, *Technical Report MS-CIS-85-18*, Department of Computer Science and Information Science, University of Pennsylvania, April, 1985.

[Kro1987] A. Kroch, Unbounded Dependencies and Subjacency in a Tree Adjoining Grammar, In A. Manaster-Ramer, editor, *Mathematics of Language*, John Benjamins, Amsterdam, 1987.

[Kub2003] J. Kubalik,J. Koutnik, and L.J. M. Rothkrantz, Grammatical Evolution with Bidirectional Representation, *Proceedings of the 6th European Conference on Genetic Programming (EuroGP 2003)*, LNCS 2610, 359-368, Springer-Verlag, 2003.

[Lan1995] K.J. Lang, Hill Climbing beats Genetic Search on a Boolean Circuit Synthesis of Koza's, *Proceedings of the 12th International Conference on Machine Learning*, Morgan Kaufmann, 1995.

[Lan1998] W.B. Langdon, *Genetic Programming + Data Structure = Automatic Programming*, Kluwer Academic, 1998.

[Lan2000b] , W.B. Langdon, Size Fair and Homologous Tree Genetic Programming Crossover, *Genetic Programming and Evolvable Machines*, **1**, 95-119, April, 2000.

[Lan2002] W.B. Langdon and R. Poli, *Foundations of Genetic Programming*, Springer-Verlag, Germany, 2002.

[Leh2003] P.K. Lehre and P.C. Haddow, Developmental Mapping and Phenotypic Complexity, *Proceedings of Congress on Evolutionary Computation (CEC 2003)*, IEEE Press, 62-65, 2003.

[Lin1973] S. Lin and B.W. Kernighan, An Effective Heuristic Algorithm for the Travelling Salesman Problem, *Operation Research*, **21**, 498-516, 1973.

[Lip1991] M. Lipsitch, Adaptation on Rugged Landscapes Generated by Iterated Local Interactions of Neighbouring Genes, *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, 1991.

[Lu1979] S.Y. Lu, The Tree-to-Tree Distance and Its Application in Cluster Analysis, *IEEE Transaction on PAMI*, **1**(2), 219-222, 1979.

[Luk2001] S. Luke S. and L. Panait, A Survey and Comparison of Tree Generation Algorithms, *Proceedings of The Genetic and Evolutionary Computation Conference (GECCO 2001)*, Morgan Kaufman Publishers, 81-88, 2001.

[Mac2003] R. M. MacCallum, Introducing a Perl Genetic Programming System: and Can Meta-evolution Solve the Bloat Problem?, *Proceedings of the 6th European Conference on Genetic Programming (EuroGP 2003)*, LNCS 2610, Springer-Verlag, 369-378, 2003.

[Man1991] B.W. Manderick and P. Spiessens, The Genetic Algorithms and the Structure of Fitness Landscape, *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, 1991.

[Man1999] C.D. Manning and H. Schtze, *Foundations of Statistical Natural Language Processing*, The MIT Press, 1999.

[MKa95] Ben McKay, M.J. Willis, and G.W. Barton, Using a Tree Structured Genetic Algorithm to Perform Symbolic Regression. *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA*, **414**, pp 487-492.

[Mit1996] M. Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, 1996.

[Mit1997] T.M. Mitchell, *Machine Learning*, McGraw-Hill, 1997.

[Miz1994] J. Mizoguchi et al, Production Genetic Algorithms for Automated Hardware Design through Evolutionary Process, *Proceedings of the First IEEE Conference on Evolutionary Computation*, IEEE Press, 85-90, 1994.

[Mol1988] R.N. Moll et al, *An Introduction to Formal Language Theory*, Springer-Verlag, Germany, 1988.

[Mon1994] D.J. Montana, Strongly-typed Genetic Programming, Technical Report BBN 7866, Bolt Beranek and Newman Inc., Cambridge, MA, 1994.

[Mon1995] D.J. Montana, Strongly-Typed Genetic Programming, *Evolutionary Computation*, **3**(2), 199-230, 1995.

[NXH2001a] Nguyen Xuan Hoai, Solving the Symbolic Regression Problem with Tree Adjunct Grammar Guided Genetic Programming: The Preliminary Results, *Proceedings of 5th Australasia-Japan Workshop in Evolutionary and Intelligent Systems*, 52-61, 2001.

[NXH2001b] Nguyen Xuan Hoai, Solving Trigonometric Identities with Tree Adjunct Grammar Guided Genetic Programming, *Proceedings of The First International Workshop on Hybrid Intelligent Systems (HIS 01)*, Physica-Verlag, 339-352, 2001.

[NXH2001c] Nguyen Xuan Hoai et al, Swarm Relational Learning: A New Leaning Paradigm, Technical Report CS5/01, Australian Defence Force Academy, University of New South Wales, 2001.

[NXH2002a] Nguyen Xuan Hoai et al, Solving the Symbolic Regression Problem with Tree Adjunct Grammar Guided Genetic Programming: The Comparative Result, *Proceedings of Congress on Evolutionary Computation (CEC 2002)*, IEEE Press, 1326-1331, 2002.

[NXH2002b] Nguyen Xuan Hoai et al, Some Experimental Results with Tree Adjunct Grammar Guided Genetic Programming, *Proceedings of the Fifth European Conference on Genetic Programming*, LNCS 2278, Springer-Verlag, 328-337, 2002.

[NXH2002c] Nguyen Xuan Hoai et al, Solving the Symbolic Regression Problem with Tree Adjunct Grammar Guided Genetic Programming, *Australian Journal of Intelligent Information Processing Systems*, **7**(3), 114-121, 2002.

[NXH2002d] Nguyen Xuan Hoai et al, Is Ambiguity is Useful or Problematic for Genetic Programming? A Case Study, *Proceedings of 4th Asia-Pacific Conference on Evolutionary Computation and Simulated Learning (SEAL 02)*, IEEE Press, 449-453, 2002.

[NXH2002e] Nguyen Xuan Hoai et al, Can Tree-Adjunct Grammar Guided Genetic Programming be Good at Finding a Needle in a Haystack? A Case Study, *Proceedings of IEEE International Conference on Communication, Circuits, and Systems*, IEEE Press, 1113-1117, 2002.

[NXH2003] Nguyen Xuan Hoai et al, Tree Adjoining Grammars, Language Bias, and Genetic Programming, *Proceedings of the 6th European Conference on Genetic Programming (EuroGP 2003)*, LNCS 2610, Springer-Verlag, 335-344, 2003.

[NXH2004a] Nguyen Xuan Hoai et al, Toward an Alternative Comparison between Different Genetic Programming System, *Proceedings of the 7th European Conference on Genetic programming (EuroGP 2004)*, LNCS3003, Springer-Verlag, 67-77, 2004.

[NXH2004b] Nguyen Xuan Hoai and R.I. McKay, An Investigation on the Roles of Insertion and Deletion Operators in Tree Adjoining Grammar Guided Genetic Programming, *Proceedings of Congress on Evolutionary Computation (CEC2004)*, IEEE Press, 472-477, 2004.

[NXH2004c] Nguyen Xuan Hoai and R.I. McKay, Softening the Structural Difficulty with TAG-based Representation and Insertion/Deletion Operators, *Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2004)*, LNCS 3103, 605-616, 2004.

[NXH2004d] Nguyen Xuan Hoai et al, Genetic Transposition in Tree Adjoining Grammar-Guided Genetic Programming: The Relocation Operator, *Proceedings of the 5th International Conference on Simulated Evolution and Learning (SEAL 04)*, 2004.

[NXH2005] Nguyen Xuan Hoai et al, Genetic Transposition in Tree Adjoining Grammar-Guided Genetic Programming: The Duplication Operator, to appear in *Proceedings of the 8th European Conference on Genetic Programming (EuroGP 2005).*

[NXHOAI2006] Representation and Structural Difficulty in Genetic Programming, accepted to IEEE Journal Transaction on Evolutionary Computation, estimated publication time: 02/2006.

[Nor1995a] P. Nordin and W. Banzhaf, Complexity Compression and Evolution, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA 95)*, Morgan Kaufmann, CA, 310-317, 1995.

[Nor1995b] P. Nordin et al, Explicitly Defined Introns and Destructive Crossover in Genetic Programming, *Proceedings of the Workshop on Genetic Programming: From Theory to Real World Applications*, 6-22, 1995.

[Nor1996] P. Nordin, F. Francone, and W. Banzhaf, Explicitly Defined Introns and Destructive Crossover in Genetic Programming, in P.J. Angeline and

K.E. Kinnear Jr, editors, *Advances in Genetic Programming 2*, chapter 6, 111-134, 1996.

[Nor1997] P. Nordin, *Evolutionary Program Induction of Binary Machine Code and Its Applications*, PhD Thesis, der Unsesitat Dortmund and Fachereich Informatik, 1997.

[Ohn1970] S. Ohno, *Evolution by Duplication*, Springer-Verlag, 1970.

[O'Ne1999] M. O'Neill and C. Ryan, Evolving Multi-Line Compilable C Programs, *Proceedings of the Second European Conference on Genetic Programming (EuroGP 1999)*, LNCS 1598, Springer-Verlag, 83-92, 1999.

[O'Ne2000a] M. O'Neill and C. Ryan, Crossover in Grammatical Evolution: A Smooth Operator? Genetic Programming, *Proceedings of the Third European Conference on Genetic Programming (EuroGP 2000)*, LNCS 1802, Springer-Verlag, 149-162, 2000.

[O'Ne2000b] M. O'Neill and C. Ryan, Grammar Based Function Definition in Grammatical Evolution, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2000)*,Morgan Kaufmann, 485-490, 2000.

[O'Ne2001a] M. O'Neill et al, Crossover in Grammatical Evolution: The Search Continues, *Proceedings of the 4th European Conference on Genetic Programming (EuroGP 2001)*, LNCS 2038, Springer-Verlag, 337-347, 2001.

[O'Ne2001b] M. O'Neill and C. Ryan, Grammatical Evolution, *IEEE Transactions on Evolutionary Computation*, **5**(4), 349-358, 2001.

[O'Ne2001c] M. O'Neill et al, Grammar Defined Introns: An Investigation Into Grammars, Introns, and Bias in Grammatical Evolution, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, Morgan Kaufmann, 97-103, 2001.

[O'Ne2003] M. O'Neill and C. Ryan, *Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language*, Genetic programming, Vol. 4, Kluwer Academic Publishers, 2003.

[O'Ne2004a] M. O'Neill and C. Ryan, Grammatical Evolution by Grammatical Evolution: The Evolution of Grammar and Genetic Code, *Proceedings of the 7th European Conference on Genetic Programming (EuroGP 2004)*, LNCS 3003, Springer-Verlag, 138-149, 2004.

[O'Ne2004b] M. O'Neill et al, The Automatic Generation of Programs for Classification Problems with Grammatical Swarm, *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2004)*, IEEE Press, 104-110, 2004.

[ORe1994] U.M. OReilly and F. Oppacher, Program Search with Hierarchical Variable Length Representation: Genetic Programming, Simulated Annealing, and Hill-Climbing, in *Parallel Problem Solving from Nature (PPSN III*, Springer-Verlag, 1994.

[ORe1995] U.M. O'Reilly and F. Oppacher, The Troubling Aspects of a Building Block Hypothesis for Genetic Programming, in L. Whitley and M.D. Vose, editors, *Foundations of Genetic Algorithms 3*, Morgan Kaufmann, 73-88, 1995.

[ORe1997] U.M. O'Reilly, Using a Distance Metric on Genetic Programs to Understand Genetic Operators, *Late Breaking Papers at the 1997 Genetic Programming Conference*, 199-206, 1997.

[ORe1998] U.M. O'Reilly, The Impact of Internal Dependency in Genetic Programming, presented at Emerging Technologies Workshop, IEEE International Conference on Evolutionary Computation, London, 1998.

[O'Su2002] J. O'Sullivan and C. Ryan (HP) An investigation into the use of different search strategies with Grammatical Evolution, *Proceedings of the*

5th European Conference on Genetic Programming (EuroGP 2002), LNCS 2278, 268-277, Springer-Verlag, 2002.

[Pal1994a] C.C. Palmer and A. Kershenbaum, Two Algorithms for Finding Optimal Communication Spanning Trees, IBM Research Report RC-19394, 1994.

[Pal1994b] C.C. Palmer and A. Kershenbaum, Representing Trees in Genetic Algorithms, *Proceedings of the First IEEE Conference on Evolutionary Computation*, 379-384, 1994.

[Pal1994c] C.C. Palmer, *An Approach to a Problem in Network Design Using Genetic Algorithms*, PhD thesis, Polytechnic University, Troy, NY, 1994.

[Pat1996] N. R. Paterson and M. Livesey, Distinguishing Genotype and Phenotype in Genetic Programming, *Late Breaking Papers at the Genetic Programming 1996 Conference*, 141-150, 1996.

[Pat1997] N. Paterson and M. Liversey, Evolving Catching Algorithms in C by GP, *Proceedings of Genetic Programming 97*, MIT Press, 262-267, 1997.

[Pat2002] *Genetic programming with context-sensitive grammars*, PhD Thesis, Saint Andrew's University, 2002.

[Per1994] T. Perkis, Stack-Based Genetic Programming, *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, IEEE Press, 148-153, 1994.

[Per1980] F.C.N. Peireira and D.H.D. Warren, Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks, *Artificial Intelligence*, **13**(3), 231-278, 1980.

[Pol1997] R. Poli and B.W. Langdon, A New Schema Theory for Genetic Programming with One-point Crossover and Point Mutation, *Genetic Program-*

ming 1998: Proceedings of the Third Annual Conference, Morgan Kaufmann, 278-285, 1997.

[Pol1998] R. Poli and B.W. Langdon, Schema Theory for Genetic Programming with One-point Crossover and Point Mutation, *Evolutionary Computation*, **6**(3), 231-252, 1998.

[Pol1999] R. Poli, Parallel Distributed Genetic Programming, in D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, McGraw-Hill, chapter 27, 1999.

[Pol1990] A.D. Polimeni and H.J. Straight, *Foundations of Discrete Mathematics*, Second Edition, Brooks/Cole Publishing Company, CA, 1990.

[Pue2002] A. O. Puente et al, Automatic Composition of Music by Means of Grammatical Evolution, *Proceedings of the 2002 conference on APL*, 148-155, ACM Press, 2002.

[Put1996] J.B. Putnam, A Grammar-Based Genetic Programming Technique Applied to Music Generation, *Proceedings of the Fifth Annual Conference on Evolutionary Programming*, MIT Press, 277-286, 1996.

[Rad1997] N.J. Radcliffe, Schema Processing, in T. Back et al, editors, *Handbook of Evolutionary Computation*, Oxford University Press, **B2.5-1-10**, 1997.

[Rat2000] A. Ratle and M. Sebag, Genetic Programming and Domain Knowledge: Beyond the Limitations of Grammar-Guided Machine Discovery, *Parallel Problem Solving from Nature (PPSN 2000)*, 211-220, 2000.

[Rat2001a] A. Ratle and M. Sebag, Avoiding the bloat with Probabilistic Grammar-Guided Genetic Programming, *Proceedings of the 5th International Conference on Artificial Evolution*, LNCS 2310, Springer Verlag, 255-266, 2001.

[Rat2001b] A. Ratle and M. Sebag, Grammar-Guided Genetic Programming and Dimensional Consistency: Application to Non-parametric Identification in Mechanics, *Applied Soft Computing*, **1**(1), 105-118, 2001.

[Rat2002] A. Ratle and M. Sebag, A Novel Approach to Machine Discovery: Genetic Programming and Stochastic Grammars, *Proceedings of Twelfth International Conference on Inductive Logic Programming*, LNCS 2583, Springer Verlag, 207-222, 2002.

[Rec1973] I. Rechenberg, *Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution*, Fromman-Hozlboog Verlag, Stuttgart, 1973.

[Ree2003] C.R. Reeves and J.E. Rowe, *Genetic Algorithms: Principles and Perspectives*, Kluwer Academic Pulisher, 2003.

[Rid1996] M. Ridley, *Evolution*, Second Edition, Blackwell Science, London, 1996.

[Ron1997] S. Ronald, Robust Encoding in Genetic Algorithms: A Survey of Encoding Issues, *Proceedings of the Forth International Conference on Evolutionary Algorithms*, 43-48, 1997.

[Ros1995] J.P. Rosca and D.H. Ballard, Causality in Genetic Programming, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, Morgan Kaufmann, 1995.

[Ros1997] J.P. Rosca, Analysis of Complexity Drift in Genetic Programming, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, Morgan Kaufmann, 286-294, 1997.

[Ros2001] B.J. Ross, Logic-based Genetic Programming with Definite Clause Translation Grammars, *New Generation Computing*, **19**(4), 313-337, 2001.

[Ros2002] B.J. Ross, The Evolution of Stochastic Regular Motifs for Protein Sequences, *New Generation Computing*, **20**(2), 187-213, 2002.

[Ros1994] G.P. Roston, *A Genetic Design Methodology for Configuration Design*, PhD. Thesis, Carnegie Mellon University, 1994.

[Rot2002] F. Rothlauf, *Representations for Genetic and Evolutionary Algorithms*, Physica-Verlag, 2002.

[Rud1976] W. Rudin, *Principles of Mathematical Analysis*, Third Edition, McGraw-Hill, 1976.

[Rya1998a] C. Ryan et al, Grammatical Evolution: Evolving Programs for an Arbitrary Language,*Proceedings of the First European Workshop on Genetic Programming*, LNCS 1391, Springer-Verlag, 83-95, 1998.

[Rya1998b] C. Ryan, M. O'Neill, and J. J. Collins, Grammatical Evolution: Solving Trigonometric Identities, in *Proceedings of the 4th International Mendel Conference on Genetic Algorithms, Optimisation Problems, Fuzzy Logic, Neural Networks, Rough Sets*, 111-119, 1998.

[Rya2002a] C. Ryan et al, No Coercion and No Prohibition, A Position Independent Encoding Scheme for Evolutionary Algorithms- The Chorus System, *Proceedings of the 5th European Conference on Genetic Programming (EuroGP 2002)*, LNCS 2278, Springer-Verlag, 131-141, 2002.

[Rya2002b] C. Ryan et al, Genetic Algorithms Using Grammatical Evolution, *Proceedings of the 5th European Conference on Genetic Programming (EuroGP 2002)*, LNCS 2278, Springer-Verlag, 278-287, 2002.

[Rya2003] C. Ryan et al, On the Avoidance of Fruitless Wraps in Grammatical Evolution, *Proceedings of the International Conference on Genetic and Evolutionary Computation Conference (GECCO 2003)*, LNCS 2724, Springer-Verlag, 1752-1763, 2003.

[Rya2004] C. Ryan et al, A Competitive Building Block Hypothesis, *Proceedings of Genetic and Evolutionary Computation (GECCO 2004)*, Springer-Verlag, 654-665, 2004.

[Sar1999] A. Sarafopoulos, Automatic Generation of Affine IFS and Strongly Typed Genetic Programming, *Proceedings of the Second European Conference on Genetic Programming (EuroGP 990*, LNCS 1598, Springer-Verlag, 149-160, 1999.

[Sch1990] Y. Schabes, *Mathematical and Computational Aspects of Lexicalized Grammars*, PhD Thesis, Department of Computer and Information Science, University of Pennsylvania, 1990.

[Sch1992] Y. Schabes, Stochastic Tree-Adjoining Grammars, *Proceedings of the 15th International Conference on Computational Linguistics (COLLING 92)*, 425-432, 1992.

[Sch1993a] Y. Schabes, Lexicalized Context-Free Grammars, Technical Report TR93-01, Mitsubishi Electric Research Laboratories, Cambridge Center, 1993.

[Sch1993b] Y. Schabes and R.C. Waters, Lexicalized Context-Free Grammar: A Cubic-Time Parsable, Lexicalized Normal Form for Context-Free Grammar That Preserves Tree Structure, Technical Report TR93-04, Mitsubishi Electric Research Laboratories, Cambridge Center, 1993.

[Sch1994] Y. Schabes and S. Shieber, An Alternative Conception of Tree-Adjoining Derivation, *Computational Linguistics*, **20** (1), 91-124, 1994.

[Sch1995] Y. Schabes and R.C. Waters, Tree Insertion Grammar: A Cubic-Time Parsable Formalism that Lexicalizes Context-Free Grammar without Changing the Trees Produced, *Computational Linguistics*, **20** (1), 479-513, 1995.

[Sch1987] J. Schmidhuber, *Evolutionary Principles in Self-Referential Learning*, Diploma Thesis, Technische Universitat, Muchen, 1987.

[Shz1995] A.G. Schultz, *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning*, Physica-Verlag, 1995.

[Shw1968] H.P. Schwefel, Projekt MHD-Staustrahlrohr: Experimentelle Optimierung einer Zweiphasenduse Teil I Technischer Bericht 11.034/68, 35, AEG Forschungsinstitut, Berlin, 1968.

[Sed1996] R. Sedgewick and P. Flajolet, *An Introduction to the Analysis of Algorithms*, Addison-Wesley, 1996.

[Sen1997] B. Sendhoff et al, A Condition for the Genotype-Phenotype Mapping: Causality, *Proceedings of the 7th International Conference on Genetic Algorithms (ICGA 97)*, Morgan Kaufmann, 73-80, 1997.

[Sha1987] V. Shanker, *A Study of Tree Adjoining Grammars*, PhD. Thesis, Department of Computer and Information Science, University of Pennsylvania, 1987.

[Sha2000] M. Shackleton et al, An Investigation of Redundant Genotype-Phenotype Mappings and Their Role in Evolutionary Search, *Proceedings of the Congress on Evolutionary Computation CEC 2000*, IEEE Press, 493-500, 2000.

[Sha2004] Y. Shan et al, Grammar-Model Based Program Evolution, *Proceedings of Congress on Evolutionary Computation (CEC'2004)*, IEEE Press, 478-485, 2004.

[Sha2002] Y. Shan, Software Project Effort Estimation Using Genetic Programming, *Proceedings of International Conference on Communications Circuits and Systems*, 2002.

[Shi1999] R. Shipman, Genetic Redundancy: Desirable or Problematic for Evolutionary Search ?, *Proceedings of the 4th International Conference on Artificial Neural Networks and Genetic Algorithms*, Springer-Verlag, 1-11, 1999.

[Shi2000] R. Shipman, M. Shackleton, I. Harvey, The Use of Neutral Genotype-Phenotype Mappings for Improved Evolutionary Search, *BT Technology Journal*, **18** (4), 103-111, 2000.

[Smi1980] S.F. Smith, *A Learning System Based on Genetic Adaptive Algorithms*, University of Pittsburgh, 1980.

[Sou1999] T. Soule and J.A. Foster, Effects of Code Growth and Parsimony Pressure on Population in Genetic Programming, *Evolutionary Computation*, **6**(4), 293-309, 1999.

[Spe1998] L. Spector et al, *Advances in Genetic Programming III*, The MIT Press, 1999.

[Spe2001] L. Spector, Autoconstructive Evolution: Push, PushGP, and Pushpop, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, Morgan Kaufmann, 137-146, 2001.

[SPECI1994] Special Issue of *Computational Intelligence: An International Journal*, November, 1994, **10** (4), Devoted to Tree Adjoing Grammars.

[Sta1992a] P.F. Stadler and W. Schnabl, The Landscape of the Travelling Salesman Problem, *Physics Letters*, **161**, 337-344, 1992.

[Sta1992b] P.F. Stadler, Correlation in Landscapes of Combinatorial Optimization Problems, *European Physics Letters*, **20**(6), 479-482, 1992.

[Sta1992c] Correlation Structure of the Landscape of the Graph-bipartioning Problem, *Journal of Physics*, **25**, 3103-3110, 1992.

[Sta2000] P. Stagge and C. Igel, Structure Optimization and Isomorphisms, *Theoretical Aspects of Evolutionary Computing*, Springer-Verlag, 2000.

[Ste1993] P.A. Stefanski, Genetic Programming Using Abstract Syntax Trees. Notes from the Genetic Programming Workshop (ICGA 93), 1993.

[Tan2003a] I. Tanev, DOM/XML-Based Portable Genetic Representation of Morphology, Behavior and Communication Abilities of Evolvable Agents, *Proceedings of the 8th International Symposium on Artificial Life and Robotics (AROB 03)*, 185-188, 2003.

[Tan2003b] I. Tanev and K. Shimohara, On the Role of Implicit Interaction and Explicit Communications in Emergence of Social Behaviour in Continuous Predators-Prey Pursuit Problem, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003)*, Springer-Verlag, 74-85, 2003.

[Tan2003c] I. Tanev and M. Brozozowski, The Effect of Explicit Communications on the Generality and Robustness of Evolved Team of Agents in Pretador-Prey Problem, *Proceedings of the First Asian-Pacific Workshop on Genetic Programming (ASPGP 2003*, 31-37.

[Tel1996] A. Teller and M. Veloso, PADO: A New Learning Architecture for Object Recognition, in K. Ikeuchi and M. Veloso, editors, *Symbolic Visual Learning*, Oxford University Press, 81-116, 1996.

[Teo2003] J. Teo, *Pareto Multi-Objective Evolution of Legged Embodied Organisms*, DIT Thesis, University of New South Wales, Australia, 2003.

[Tsa2002] A.D. Tsakonas and G. Dounias, A Scheme for the Evolution of Feedforward Neural Networks using BNF-Grammar Driven Genetic Programming, *Proceedings of European Symposium on Intelligent Technologies, Hybrid Systems and Their Implementation on Smart Adaptive Systems*, 115-121, 2002.

[Van2003a] L. Vanneschi et al, Fitness Distance Correlation in Structural Mutation Genetic Programming, in *Proceedings of the 6th European Conference on Genetic Programming (Euro 2003)*, LNCS 2610, Springer-Verlag, 455-464, 2003.

[Van2003b] L. Vanneschi et al, Fitness Distance Correlation in Genetic Programming: a Constructive Counterexample, *Proceedings of Congress on Evolutionary Computation (CEC 2003)*, IEEE Press, 289-296, 2003.

[Vas1999] V. K. Vassilev et al, Digital Circuit Evolution and Fitness Landscapes, *Proceedings of the Congress on Evolutionary Computation*, IEEE Press, 1999.

[Vas2000] V.K. Vassilev et al, Information Characteristics and the Structure of Landscapes, *Evolutionary Computation*, **8**(1), 31-60, 2000.

[Vas2003] V. K. Vassilev et al, Smoothness, Ruggedness and Neutrality of Fitness Landscapes: from Theory to Application, *Advances in Evolutionary Computing: Theory and Applications*, Springer-Verlag, 3-44, 2003.

[Vos1999] N.M. Vose, *The Simple Genetic Algorithm: Foundations and Theory*, MIT Press, 1999.

[Wat1995] M.S. Waterman, *Introduction to Computational Biology: Maps, Sequences, and Genomes*, Chapman & Hall, 1995.

[Weh1990] K.H. Wehrhahn, *Combinatorics: An Introduction*, Carslaw Publications, Sydney, 1990

[Wei1990] E.D. Weinberger, Correlated and Uncorrelated Landscapes and How to Tell the Difference, *Biological Cybernetics*, **63**, 325-336, 1990.

[Wei1991] E.D. Weinberger, Local Properties of Kauffman's N-k Model: A Tunably Rugged Energy Landscape, *Physical Reviews*, **44**, 63-99, 1991.

[Wei1988] D.J. Weir, *Characterizing Mildly Context-Sensitive Grammar Formalisms*, PhD. Thesis, Department of Computer and Information Science, University of Pennsylvania, 1988.

[Whi1994] P.A. Whigham, Context-Free Grammar and Genetic Programming, Technical Report CS20/94, Department of Computer Science, ADFA, University of New South Wales, Australia, 1994.

[Whi1995a] P.A. Whigham, Grammatically-Based Genetic Programming, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, Morgan Kaufmann, 33-41, 1995.

[Whi1995b] P.A. Whigham, Inductive Bias and Genetic Programming, *Proceedings of the First International Conference on Genetic Algorithms in Engineering System: Innovation and Applications*, 461-466, 1995.

[Whi1995c] P.A. Whigham, A Schema Theorem for Context-Free Grammars, *Proceedings of the 1995 IEEE International Conference on Evolutionary Computation*, IEEE Press, 178-182, 1995.

[Whi1996] P.A. Whigham, *Grammatical Bias for Evolutionary Learning*, PhD thesis, University of New South Wales, 1996.

[Whi2001a] P. A. Whigham and P. F. Crapper, Modelling Rainfall-runoff Using Genetic programming, *Mathematical and Computer Modelling*, **33**(6-7), 707-721, 2001.

[Whi2001b] P.A. Whigham and F. Recknagel, An Inductive Approach to Ecological Time Series Modelling by Evolutionary computation, *Ecological Modelling*, **146**(1-3), 275-287, 2001.

[Won1994] M.L. Wong and K.S. Leung, Learning First-order Relations from Noisy Databases using Genetic Algorithms, *Proceedings of the Singapore Second International Conference on Intelligent Systems*, 159-164, 1994.

[Won1995a] M.L. Wong and K.S. Leung, An Adaptive Inductive Logic Programming System using Genetic Programming, *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, MIT Press, 737-752, 1995.

[Won1995b] M.L. Wong and K.S. Leung, Combining Genetic Programming and Inductive Logic Programming using Logic Grammars, *Proceedings of the IEEE International Conference on Evolutionary Computing*, IEEE Press, 733-736, 1995.

[Wol1996] D.H. Wolpert and W.G. Macready, No Free Lunch Theorems for Search, technical report SFI-TR-95-02-010, Santa Fe, 1996.

[Won1995c] M.L. Wong and K.S. Leung, Applying Logic Grammars to Induce sub- functions in Genetic Programming, *Proceedings of the IEEE International Conference on Evolutionary Computing*, IEEE Press, 737- 740, 1995.

[Won1995d] M.L. Wong and K.S. Leung, Genetic Logic Programming and Applications, *IEEE Expert*, **10**(5), 68-76, 1995.

[Won1996] M.L. Wong and K.S. Leung, Evolving Recursive Functions for the Even-parity Problem Using Genetic Programming, in P. J. Angeline and K. E. Kinnear Jr, editors, *Advances in Genetic Programming 2*, MIT Press, 221- 240. 1996.

[Won1997] M.L. Wong and K.S. Leung, Evolutionary Program Induction Directed by Logic Grammars, *Evolutionary Computation*, **5**(2), 143-180, 1997.

[Won2000] M.L. Wong and K.S. Leung, *Data Mining Using Grammar Based Genetic Programming and Applications*, Kluwer Academic Publishers, 2000.

[Won2003] M.L. Wong, Personal Communication, Dec 2003.

[Wri1932] S. Wright, The Roles of Mutation, Inbreeding, Crossbreeding, and Selection in Evolution, Proceedings of the 6th International Congress on Genetics,**1**, 356-366, 1932.

[XTA1995] XTAG-Group, A Lexicalised Tree Adjoining Grammar of English, Technical Report, Institute for Research in Cognitive Science (IRCS), University of Pennsylvania, 95-03, 1995.

[Yok1995] T. Yokomori and S. Kobayashi, DNA Evolutionary Linguistics and RNA Structure Modelling: A Computational Approach, *Proceedings of the First International Symposium on Intelligence in Neural and Biological Systems*, IEEE Press, 38-45, 1995.

[Zha1995] B.T. Zhang and H. Muhlenbein, Balancing Accuracy and Parsimony in Genetic Programming, *Evolutionary Computation*, **3**(1), 17-38, 1995.

[Zit1999] E. Zitzler and L. Thiele, Multi-objective Evolutionary Algorithms: A Comparative Case Study and The Strength Pareto Approach, *IEEE Trans on Evolutionary Computation*, **3**(1), 257-271, 1999.

[Zit2001] E. Zitzler et al, SPEA2: Improving the Strength Pareto Evolutionary Algorithm, Technical Report 103, Computer Engineering and Networks Laboratory (TK), ETH Zurich, Switzerland, 2001.

[Zva2004] S. Zvada and R. Vanyi, Improving Grammar Based Evolution Algorithms via Attributed Derivation Trees, *Proceedings of the 7th European Conference on Genetic Programming (EuroGP 2004)*, LNCS 3003, Springer-Verlag, 208-219, 2004.

Appendix A

Some More Information on Grammars

In this appendix, we briefly revise some important concepts relating to grammars and their derivation trees, as used in grammar guided genetic programming. Apart from the TAGs used for representation in this thesis, as pointed out in chapter 2, there are four main classes of grammars which have been used in the grammar guided genetic programming literature, namely context-free grammars (CFGs), attribute grammars (AGs), definite clause grammars (DCGs), and definite clause translation grammar (DCTGs), so all are covered below. The appendix concludes with grammars for some of the problems investigated in the thesis.

A.1 Context Free Grammars

A context free grammar (CFG) ([Hop1979, Mol1988]) is a four-tuple $G = \sum, N, P, S)$ where N is a set of nonterminal symbols, $\sum$ is a set of terminal symbols, with $\sum \cap N = \emptyset$, $S \in N$ is the starting symbol, and, P is the set of production rules. Each rule $p \in P$ is of the form $A \to x$, where A is a nonterminal symbol ($A \in N$) and x is a string composed of nonterminal and terminal symbols ($x \in (N \cup \sum)*$). In the Chomsky hierachy of formalisms [Cho1956], CFGs are the most popular, being widely used in various sciences. The following example depicts a simple context free grammar for generating arithmetic expressions of two variable (x and y)

An example of context free grammars. $G = (\sum, N, P, EXP)$, where $N = \{EXP\}$, $\sum = \{+, -, *, \, x, y\}$ and P is as follows:

$EXP \rightarrow EXP + EXP$

$EXP \rightarrow EXP - EXP$

$EXP \rightarrow EXP * EXP$

$EXP \rightarrow EXP/EXP$

$EXP \rightarrow x$

$EXP \rightarrow y$

The languages resulting from CFGs are defined by means of string-rewriting operations called derivation steps. A derivation step is the application of rewriting a nonterminal symbol A in a string, by replacing with the right hand side of one of the rules that have A as the left hand side. In symbolic form, if $S = xAy$ (where $x, y \in (\sum \cup N)^*$) is a string and $A \rightarrow \alpha$ is a rule in P, then S can be rewritten as $S = x\alpha y$. The derivation step is denoted as $xAy \overset{A \rightarrow \alpha}{\Rightarrow} x\alpha y$. The string language L of a CFG $G = (N, \sum, P, S)$ is defined as

$L_G = \{x \in (\sum \cup N)^* \ /S \overset{*}{\Rightarrow} x\}$

where $\overset{*}{\Rightarrow}$ is the finite transitive closure of $\Rightarrow$ (i.e it is a finite sequence of $\Rightarrow$). For each $x \in L_G$, there is a finite sequence of derivation step for deriving x. This derivation sequence can be represented by a tree, called the derivation tree, built as follows. The derivation tree starts with the root S. Going through the derivation sequence from left to right if there is a derivation step $A \Rightarrow \alpha$, then scan through the derivation tree in top-down and from left-to-right fashion to pick up the first leaf node labelled with A. Extend the derivation tree at that node by creating its child nodes, and label them with the corresponding symbols appearing in α. Figure A.1 depicts an example of a derivation sequence and its corresponding derivation tree.

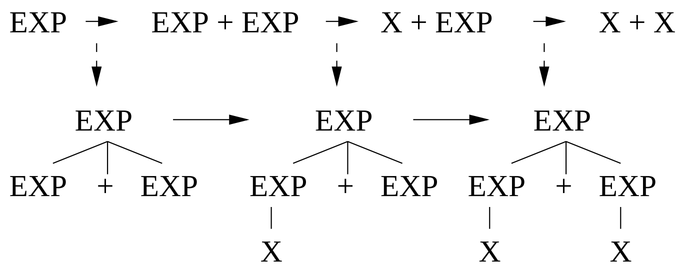


Figure A.1: An example of CFG derivation sequences and trees

A.2 Attribute Grammars

Attribute grammars are perhaps the most popular among the many extensions to context free grammars which augmenting CFGs to handle some level of context-dependence. They have been widely used to represent the semantics of programs in compilers [Aho1986]. They were proposed by Knuth in his seminal paper [Knu1968]. In this paper, an attribute grammar is defined as a pair consisting of a context free grammar(CFG) and a an attribute system. Terminal strings are processed in two steps. First, the CFG is used to parse the terminal string. Then, the nodes in the parse tree are decorated with attribute name/value pairs. The attribute system associates each CFG rule with a set of semantic rules, which specify the dependencies between the attributes of a parent node and the attribute of its children when the parent's nonterminal is expanded via that CFG rule.

There is no concept of derivation tree for attribute grammars. Instead they used the semantic rules to assign sets of attributes values to the nodes of CFG derivation trees. Different types of attribute grammars have different ways of assigning and evaluating attribute values. [Der1988] provides a comprehensive review of different types of attribute grammars.

A.3 Definite Clause Grammars (DCGs)

Definite Clause Grammars, proposed in [Per1980], are special form of logic grammars [Abr1989]. They are based on PROLOG developed by A. Colmerauer in the mid-70s. A DCG is very similar to an attribute grammar, as it is defined as a generalisation of a context free grammar in which the nonterminal and terminal symbols are allowed to have arguments. The arguments play a similar role to the attributes in attribute grammars. The only different between definite clause grammars and attribute grammars are just the notations. Since DCGs are derived from PROLOG, their notation are heavily dependent on PROLOG syntax. So a rule which includes arguments (attributes) for the symbols might look like: $VerbPhrase(E) \rightarrow TransitiveVerb, Sent(E).$. Similar to what happens in attribute grammars, the derivation trees of DCGs are essentially the derivation trees of CFGs further decorated with arguments. In fact, the arguments represent a fragment of a PROLOG program. More details on DCGs can be found in [Abr1989].

A.4 Definite Clause Trabslation Grammars

Proposed in citeAbramson1984, definite clause translation grammars (DCTG) are an extension of DCGs, providing increased semantic expressiveness by allowing a set of semantic rules, implemented as horn clauses, to be attached to each nonterminal node in the derivation tree. These rules guide the computation of semantic properties of the node in terms of the semantic properties of the subtrees of the node. Futher information on DCTG could be found in [Abr1989].

A.5 Schabes's Lexicalisation Algorithm

This section describes the algorithm in [Sch1990, Jos1997] for finding a LTAG G_{lex} that strongly lexicalizes a given context free grammar G. In other words, the derived tree set of G_{lex} is the same as the derivation tree set of G. The main idea of the algorithm is to separate the recursive part of the grammar G from the non-recursive part. The non-recursive part generates a finite number of trees. We use them as the initial (α) trees in our G_{lex}. Whenever G contains a recursion of the form $B \overset{*}{\Rightarrow} \alpha B\beta$, we create a set of B-type auxiliary trees in which α and β are expanded, in all possible ways, by the non-recursive part of the grammar.

The preprocessing of the algorithm is therefore to separate the recursive part of G from the non-recursive part. To accomplish that task, we need to find out the recursive symbols and recursive rules in G. They are defined as

Definition (recursive symbols and rules). *Let $G = (\sum, N, P, S)$ be a finitely ambiguous grammar (i.e there is no B for which $B \overset{*}{\Rightarrow} B$ for any $B \in N$), and suppose that λ (empty word) is not in the language of G, $L(G)$. We say that $B \in N$ is a recursive symbol if and only if $\exists \alpha, \beta \in (\sum \cup N)^*$ such that $B \overset{*}{\Rightarrow} \alpha B\beta$. We say that a production rule $B \rightarrow \delta$ is recursive whenever B is recursive through this rule.*

The next step is to partition the rule set P of G into two sets: the set of recursive rules, $R \subseteq P$, and the set of non-recursive rules, $NR \subseteq P$, satisfying $R \cup NR = P$ and $R \cap NR = \emptyset$. In order to determine whether a rule is recursive, a directed graph G_P is constructed as follows. All the nodes of G_P are nonterminal symbols in G, and each edge of G_P is labelled by a production rule in P. For two nodes B and C in G_P, there is an edge labelled with p from B to C if there is a rule $p \in P$ of the form $B \rightarrow \alpha C\beta$, where α and $\beta \in (\sum \cup N)^*$. Then, a symbol B is recursive if node B belongs to a cycle in G_P. A rule $p \in P$ is recursive if there is an edge labelled with p which belongs to a cycle.

The algorithm for finding a LTAG G_{lex} for each CFG G is as follows:

```
1)  Extract all derivation trees of $G$ using the rules
    in $NR$ only.
2)  Set the set of initial trees, $I$, of $G_{lex}$ as the set
    from step 1.
3)  Find the base cycles $\{c_1...c_k\}$ in $G_P$ using classical
    algorithms such as those in [Gib1985].
4)  Set the set of auxiliary trees of $G_{lex}$, $A = \emptyset$.
5)  Repeat procedure CreateAux given below for all $c_i$ $(i = 1...k)$
    until no more trees can be added to A.
```

```
PROCEDURE CreateAux($c_i$)
 FOR all nodes $n_i$ in $c_i$, let $B_i$ be the label of $n_i$,
 According to $c_i$, $B_i \Rightarrow \overset{*}{\alpha_i}\beta_i$,
 IF $B_i$ is the label of a node in a tree in $I \cup A$ THEN
  FOR all derivation $\alpha_i \overset{*}{\Rightarrow} w_i \in \sum*$, $\beta_i \overset{*}{\Rightarrow} z_i \in \sum*$
  that use only rules in $NR$
   Add to $A$ the auxiliary tree correspoinding to all derivations:
   $B_i \overset{*}{\Rightarrow} \alpha_i B_i \beta_i \overset{*}{\Rightarrow} w_i B_i z_i$ where the node labelled $B_i$ on the frontier
   is the foot node.
```

A.6 The Grammars for Some Problems in the Thesis

This section give the details of some problem grammars used in chapter 6 and 9.

Note: as is usual in grammar representations, we use the "—" symbol to abbreviate productions with the same left hand side, i.e. the notation $A \rightarrow \beta|\gamma$ is an abbreviation for the pair of productions $A \rightarrow \beta$ and $A \rightarrow \gamma$

A.6.1 Some Grammars for the Problems in Chapter 6

Grammars for the ORDER and MAJORITY problems: $G = (\sum, N, P, S)$, where $\sum = \{Join, P_1, P_2, ..., P_n, N_1, N_2, ..., N_n\}$, $N = \{S\}$, and the rule set P is

$S \rightarrow S\ Join\ S$

$S \rightarrow P_1|P_2|...|P_n|N_1|N_2, ...|N_n$

$G_{lex} = (\sum, N, S, I, A)$, where $I \cup A$ is as in Figure A.2 with T being a lexicon that can be substituted with a member of the set $\{P_1, P_2, ..., P_n, N_1, N_2, ..., N_n\}$

Grammars for the SEXTIC and QUINTIC problems: $G = (\sum, N, P, S)$, where $\sum = \{X, +, -, *, /\}$, $N = \{EXP, OP, VAR\}$, and the rule set P is

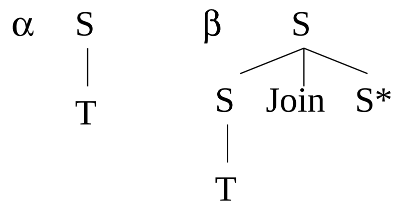


Figure A.2: Elementary trees for the grammar of the ORDER and MAJORITY problems

$EXP \rightarrow EXP\ OP\ EXP \mid VAR$

$OP \rightarrow + | - \mid * \mid /$

$VAR \rightarrow X$

$G_{lex} = (\sum, N, S, I, A)$, where $I \cup A$ is as in Figure A.3.

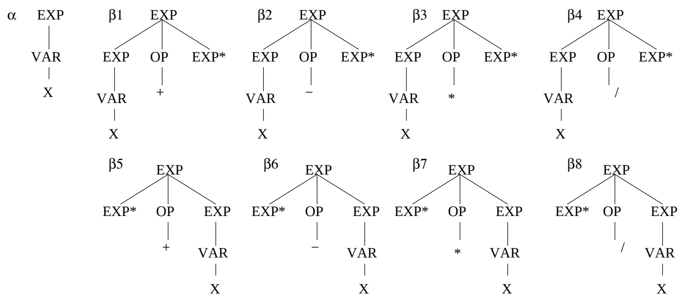

α EXP VAR X
β1 EXP EXP OP EXP* VAR + X
β2 EXP EXP OP EXP* VAR − X
β3 EXP EXP OP EXP* VAR * X
β4 EXP EXP OP EXP* VAR / X
β5 EXP EXP* OP EXP + VAR X
β6 EXP EXP* OP EXP − VAR X
β7 EXP EXP* OP EXP * VAR X
β8 EXP EXP* OP EXP / VAR X

Figure A.3: Elementary trees for the grammar of the SEXTIC and QUINTIC problems

Grammars for the trigonometric indentity (TRIGO) problem: $G = (\sum, N, P, S)$, where $\sum = \{X, 1.0, +, -, *, /, sin\}$, $N = \{EXP, OP, PREOP, VAR\}$, and the rule set P is

$EXP \rightarrow EXP\ OP\ EXP \mid PREOPEXP| \, VAR$

$OP \rightarrow + | - \mid * \mid /$

$PREOP \rightarrow sin$

$VAR \rightarrow X$

$G_{lex} = (\sum, N, S, I, A)$, where $I \cup A$ is as in Figure A.4, with T being a lexicon that can be substituted with either X or 1.0

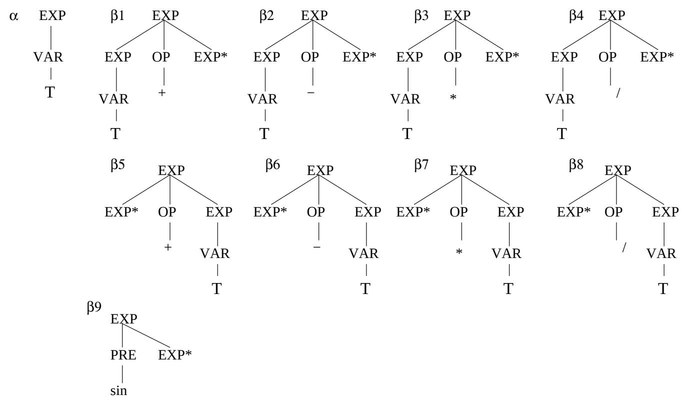


Figure A.4: Elementary trees for the grammar of the TRIGO problems

Grammars for the TWOBOX problem: $G = (\sum, N, P, S)$, where $\sum = \{W, H, L, w, h, l, +, -, *, /\}$, $N = \{EXP, OP, VAR\}$, and the rule set P is

$EXP \rightarrow EXP\ OP\ EXP \mid VAR$

$OP \rightarrow + | - | * | /$

$VAR \rightarrow W|H|L|w|h|l$

$G_{lex} = (\sum, N, S, I, A)$, where $I \cup A$ is as in Figure A.5, with T being a lexicon that can be substituted with a member of the set $\{W, H, L, w, h, l\}$.

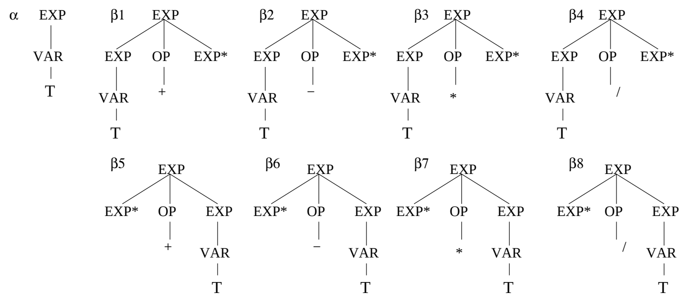


Figure A.5: Elementary trees for the grammar of the TWOBOX problem

A.6.2 Some Grammars for the Problems in Chapter 9

The corresponding TAGs G_{lex} of the four grammars given for the 6-multiplexer are as follows

Corresponding TAG for G_1: $G1_{lex} = \{\sum, N, I, A, B\}$, where $\sum$ and N are the same as in G_1, the elementary tree set $E = A \cup I$ is depicted in Figure A.6, where TL stands for a lexicon that can be substituted with any of $a_0, a_1, ..., d_3$.

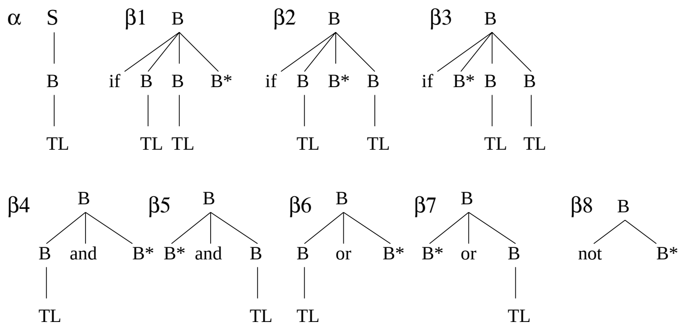


Figure A.6: TAG elementary trees for $G1_{lex}$

Corresponding TAG for G_2: $G2_{lex} = \{\sum, N, I, A, B\}$, where $\sum$ and N are the same as in G_2, the elementary tree set $E = A \cup I$ is depicted in Figure A.7, where ADD is a lexicon that can be substituted with either a_0 or a_1; and T L stands for a lexicon that can be substituted with any of $a_0, a_1, ..., d_3$.

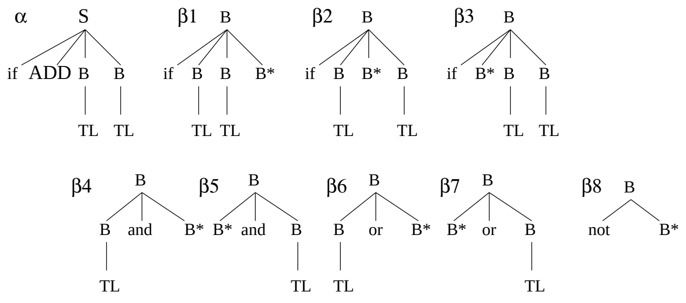


Figure A.7: TAG elementary trees for $G2_{lex}$

Corresponding TAG for G_3: $G3_{lex} = \{\sum, N, I, A, B\}$, where $\sum$ and N are the same as in G_3, the elementary tree set $E = A \cup I$ is depicted in Figure A.8, where ADD is a lexicon that can be substituted with either a_0 or a_1; and TL stands for a lexicon that can be substituted with any of $a_0, a_1, ..., d_3$.

Corresponding TAG for G_4: $G4_{lex} = \{\sum, N, I, A, B\}$, where $\sum$ and N are the same as in G_4, the elementary tree set $E = A \cup I$ is depicted in Figure A.9,

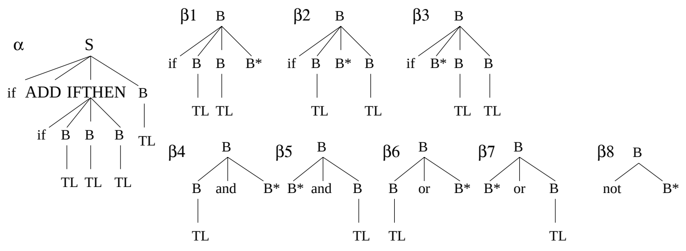


Figure A.8: TAG elementary trees for $G3_{lex}$

where TL stands for a lexicon that can be substituted with either of a_0, a_1,...,d_3.

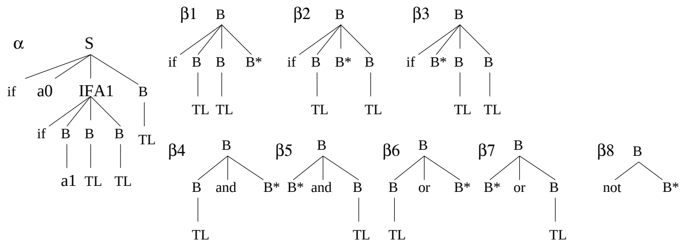


Figure A.9: TAG elementary trees for $G4_{lex}$

Appendix B

Schema Theory

B.1 Introduction

This appendix gives a brief overview of some important schema theories in GA, GP, and GGGP. First, we present the concept of schema, and a schema theorem, for fixed-size binary GA representation from [Hol1975]. Next, we present the definition of fixed-size and -shape schemata, and the schema theorem, for standard GP using expression tree representation from [Pol1997, Pol1998, Lan2002]. Finally, the schemata and schema theorem for GGGP proposed by Whigham ([Whi1995c, Whi1996]) are briefly presented.

B.2 Holland's Schema Theorem for GAs

The first schema theory was proposed by Holland for GAs using fixed-length binary representation. In binary representation the content of each gene in a chromosome (string) is either of two arbitrary values. Without losing generality suppose that the alphabet set of such GAs is 0,1. A schema, as defined in [Hol1975, Gol1989], plays a role of a similarity template describing a subset of binary strings with similarities at some certain string positions. It is done by, firstly, extending the alphabet set to 0,1,*, where * is called the "don't care" symbol. Then, a schema is defined as a string (of fixed length equal to the length of each chromosome) of 0,1, and * symbols. In a schema, the positions that have "don't care" symbols can be instantiated by replacing them with either 0 or 1. Thus, a schema represents a set of binary strings. For instance, schema *1**0 represent 8 strings, and 11000 is one of them. 11000 is said to be matched with schema *1**0.

Each schema H is characterised by two properties, namely, the schema defining length $\delta(H)$ and the schema order $o(H)$. The schema defining length is the distance between the two furthest non-* symbols, whereas the schema order is the number of non-* symbols in the schema. For the schema H=*1**0, $\delta(H) = 4$ and $o(H) = 2$.

The schema theorem for GAs with binary representation, fitness-proportionate selection, 1-point crossover, and 1-point mutation is stated by the following equation [Hol1975, Gol1989]:

$$E[m(H, t+1)] \geq m(H,t)\frac{f(H,t)}{\overline{f}(t)}(1-p_m)^{o(H)}\left[1 - p_c\frac{\delta(H)}{N-1}(1 - \frac{m(H,t)f(H,t)}{M\overline{f}(t)})\right] \tag{B.1}$$

where

$m(H,t)$ is the number of strings matching schema H at generation t.

$f(H,t)$ is the mean fitness of the strings matching schema H.

$\overline{f}(t)$ is the mean fitness of all strings in the population.

p_m is the probability per bit of the mutation operator.

p_c is the probability of the crossover operator.

N is the number of bits in each string (chromosome).

M is the number of strings in the population (population size).

$E[m(H, t+1)\ ]$ is the expected number of strings matching the schema H at generation $t + 1$.

Holland's schema theorem above describes the lower bound (pessimistic number) for the number of strings matching a schema H in the next generation. The theorem is used to explain that schemata which have high average fitness, short defining length, and low order, will survive and multiply exponentially under selection pressure during the evolutionary process. Therefore, the essence of GA search is the exponential increase of parallel sampling of the promising subspaces represented by schemata [Gol1989]

B.3 Fixed Shape and Size Schema Theorem for GP by Poli and Langdon

In [Pol1997, Pol1998], Poli and Langdon extended the concept of rooted schema proposed by Rosca ([Ros1997]) to rooted schemata with fixed shape and size. A rooted fixed-shape and -size schema is defined as follows ([Lan2002]):

Definition B.1 (GP fixed shape and size Schemata). *A GP schema is a rooted tree composed of nodes from the set $F \cup T \cup \{=\}$, where F and T are the sets of functions and terminals respectively. The operator "=" is a polymorphic function whose arity can take any of the arities of the functions in F. The symbol "=" plays the role played by the "don't care" symbol in GA schemata. We note that the terminals in T have zero arity.*

The following figure B.1 depicts a fixed-shape and -size schema.

For the fixed-shape and -size schema H defined as above, the concepts of order, length, and defining length from GA schemata can be extended as follows.

Order The number of non-"=" symbols, written as $o(H)$

Length The total number of nodes in the schema, written as $N(H)$

Defining Length The number of links in the minimal tree fragment within the schema composed entirely of non-"=" symbols, written as $L(H)$

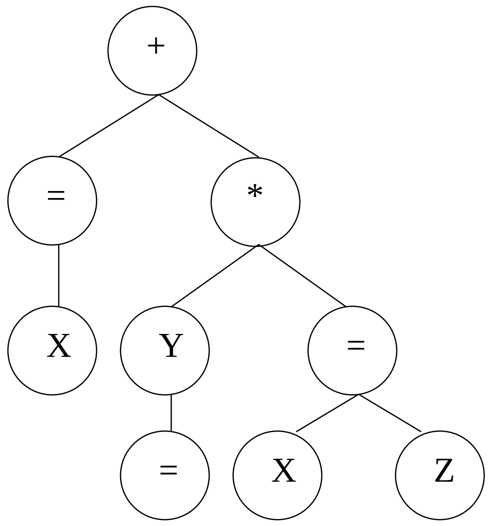


Figure B.1: An example of a fixed-shape and -size schema in GP expression tree representation

Two further important concepts, of hyperspaces and hyperplanes, are defined for fixed-shape and -size schemata as folows:

Definition B.2. (hyperspaces and hyperplanes). *A schema G is called a hyperspace if it contains the "=" symbol only. In other word, $o(G) = 0$. A schema H is called a hyperplane if it has at least one non-"=" symbol. The schema G obtained from a hyperplane H by replacing all non-"=" symbols in H with "=" symbols is known as the hyperspace associated with H.*

In [Lan2002], the genetic operators used were one-point crossover and one-point mutation. The one-point crossover between two GP expression trees is accompished by the following three steps:

1. *Alignment.* Matching the two parents top down from the roots in a recursive way, to find the identically shaped sub-parts of the two trees from the roots (i.e matching the arity of nodes visited between the two trees). The recursion stops when a mis-match in arity is detected, between two currently matching nodes. The common top part of the two trees is stored.

2. *Crossover point selection.* Choosing a crossover point at random from the

common part, with uniform probability.

3. *Swap*. Swapping the two subtrees beneath the crossover points in the two parents.

The one-point mutation operator uses the point mutation proposed in [MKa95], where the function/terminal in a randomly chosen node in the GP expression tree is replaced by a randomly chosen function/terminal with the same arity.

The formula for estimating a lower bound on the propagation of a fixed-shape and -size GP schema using fitness-propotionate selection, one-point crossover, and one-point mutation is as follows [Lan2002]

$$E[m(H,t{+}1)] \geq L\times(1{-}p_m)^{o(H)}\left\{1-p_c\left[\begin{array}{c} p_{diff}(t)\left(1-\frac{m(G(H),t)f(G(H),t)}{M\overline{f}(t)}\right) \\ + \\ \frac{L(H)}{N(H)-1}\frac{m(G(H),t)f(G(H),t)-m(H,t)f(H,t)}{M\overline{f}(t)} \end{array}\right]\right\} \tag{B.2}$$

where $L = m(H,t)\frac{f(H,t)}{\overline{f}(t)}$; the meaning of $m(H,t)$, $f(H,t)$, p_c, p_m, $\overline{f}$ are similar to those in the GA schema theorem presented in the previous section; m(G(H),t) is the number of individuals in the population at generation t that match the hyperspace associated with schema H; $f(G(H),t)$ is the average fitness of those individuals; and, p_{diff} is the probability that schema H is disrupted when an individual $h \in H$ is crossed over with an individual $\overline{h} \notin G(H)$.

B.4 Whigham's Schema Theorem for CFG-GP

In [Whi1995c, Whi1996], a schema theory was derived for context-free grammar guided genetic programming. The concept of schema is based on partial derivation trees of the grammar. It is defined as follows [Whi1995c]

Definition B.3 (CFG-GP schema). *A schema H for a context-free grammar based representation is the partial derivation tree $A \rightarrow *\theta$, where $A \in N$ (the nonterminal symbol set) and $\theta \in \{N \cup \sum\}^*$.*

Figure B.2 depicts a schema in CFG-GP.

In [Whi1996], a simple schema theorem for the propagation of CFGGP schemata

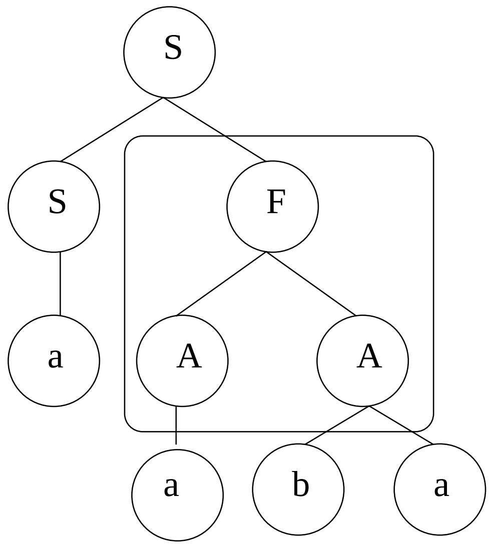


Figure B.2: An example for a CFG-GP schema. The Schema is $F \Rightarrow A\ A$

was derived as

$$E[m(H,t+1)] \geq m(H,t)\frac{f(H,t)}{\bar(f)(t)} \times \{(1-p_m\overline{P}_{d_m}(H,t))(1-p_c\overline{P}_{d_c}(H,t)\} \quad \text{(B.3)}$$

where the meanings of $m(H,t)$, $f(H,t)$, p_c, p_m, $\overline{f}$ are similar to those in previous sections; $\overline{P}_{d_m}(H,t)$ and $\overline{P}_{d_c}(H,t)$ are the average disruptions at generation t of schema H due to crossover and mutation repsectively. More details on how to calculate $\overline{P}_{d_m}(H,t)$ and $\overline{P}_{d_c}(H,t)$ can be found in [Whi1996] (chapter 6).

Appendix C

Techniques for Fitness Landscape Analysis

In this appendix, we briefly overview two major techniques for analysing fitness landscapes based on random walks namely correlation analysis and information content measures [Deb1997, Ree2003, Teo2003]. The former uses statistical measures to estimate the smoothness of fitness landscapes, by examining how the fitness values of neighbouring points correlate with each other; the latter employs an information metric to investigate the ruggedness and modality of fitness landscapes.

C.1 Correlation Analysis of Fitness Landscapes

Correlation analysis is a set of statistical techniques used to characterize the difficulty of search problems by measuring the correlation among the fitnesses of neighbouring points, and the correlation between parents and offspring. There are three main techniques of correlation analysis, using the autocorrelation metric of a fitness landscape, the fitness correlation coefficients of genetic operators, and fitness-distance correlation [Deb1997]. The first has been used to study fitness landscapes in the field of evolutionary biology and evolutionary algorithms, and in this thesis. T details of this technique is given here. Descriptions of the others can be found in [Deb1997], [Man1991], and [Jon1995c].

The autocorrelation function of a random walk in the landscape appears to have been the first tool used to analyse fitness landscapes [Wei1990]. For a given fitness landscape $((S, d), f)$, where (S, d) is the metric space and f is a fitness function, select a random starting point s_0 . Then select a random neighbour s_1 of s_0 (i.e. $d(s_0, s_1) = 1$). Repeat this process N times, collecting the fitness sequence $F = \{f(s_i)\}_{i=0}^{N}$. The autocorrelation function ρ of the random walk is defined as follows:

$$\rho(h) = \frac{R(h)}{s_f^2} \tag{C.1}$$

Where s_f^2 is the variance of the fitness sequence, $s_f^2 = \sum_{i=0}^{N}(f(s_i) - m_F)^2$; and $R(h)$ is the autocovariance function of the fitness sequence F. For each h, $R(h)$ is estimated by:

$$R(h) = \frac{1}{N}\sum_{i=0}^{N-h}(f(s_i) - m_F)(f(s_i) - m_F) \tag{C.2}$$

where $m_F = \frac{1}{N+1}\sum_{i=0}^{N} f(s_i)$ is the mean of the fitness sequence F. The autocorrelation function $\rho(h)$ expresses, for each distance h, the level of correlation between search points a distance h apart. A second measure derived from the autocorrelation function is the correlation length, usually denoted as τ, defined as the distance h where $\rho(h) = 1/2$. The larger the correlation length τ, the more correlated and smoother is the fitness landscape. Small τ indicates rugged fitness landscapes. The autocorrelation ρ and the correlation length τ provide a rough indication of how difficult a fitness landscape is. They were used in characterising the fitness landscapes of a number of well known problem families and fitness landscape models ([Lip1991], [Man1991], [Wei1991], [Sta1992b], and [Sta1992c]).

C.2 Information Content Measures for Fitness Landscape

[Vas1999, Vas2000] proposed a methodology based on the concept of information content ([Abr1963, Cov1991]) to analyse fitness landscapes. From an N-step random walk to neighbouring points, a sequence of fitness values $\{f_t\}_{t=1}^{N}$ is obtained. Four measures of the entropy and amount of fitness change encountered during the walk can be calculated:

1. *Information Content* ($H(\epsilon)$): indicates the ruggedness of landscape.
2. *Partial Information Content* ($M(\epsilon)$): indicates the modality of landscape.
3. *Information Stability* ($\epsilon*$): indicates the magnitude of optima in landscape.
4. *Density-Basin Information* ($h(\epsilon)$): characterizes the landscape structure around optima.

The information content characterizes the amount of ruggedness in the flat areas of the fitness landscape. It is defined as follows:

$$H(\epsilon) = -\sum_{p \neq q} P_{[pq]} log_6 P_{[pq]} \tag{C.3}$$

The probability $P_{[pq]}$ represents the frequencies of all 6 possible pairs pq in the string $S(\epsilon) = s_1 s_2 s_3 ... s_N$ where $s_i \in \{\overline{1}, 0, 1\}$. The string $S(\epsilon)$ indicates the changes of fitness landscape values with ϵ-tolerance. Each s_i in $S(\epsilon)$ is defined as

$$s_i = \Psi_{f_t}(i, \epsilon) \tag{C.4}$$

where

$$\Psi_{f_t}(i, \epsilon) = \begin{cases} \overline{1} & \text{if } f_i - f_{i-1} < -\epsilon \\ 0 & \text{if } |f_i - f_{i-1}| \leq \epsilon \\ 1 & \text{if } f_i - f_{i-1} > \epsilon \end{cases} \tag{C.5}$$

In other words, the information content $H(\epsilon)$ measures the entropy of the different directions of change encountered during the random walk, the parameter ϵ setting a threshold for the sensitivity of the entropy measurement. ϵ is a real number from the range [0..L], where L is the maximal pair-wise difference in the sequence $\{f_t\}_{t=1}^{N}$. The information measure is most sensitive when $\epsilon = 0$, and insensitive when $\epsilon = L$.

The partial information content is defined as follows:

$$M(\epsilon) = \frac{\mu}{N} \tag{C.6}$$

where μ is the length of the derived string $S'(\epsilon)$. μ is calculated by calling a recursive function of three integer arguments and called by $\Phi_S(1, 0, 0)$. This recursive function is defined as

$$\Phi_S(i, j, k) = \begin{cases} k & \text{if } i > n \\ \Phi_S(i+1, i, k+1) & \text{if } j = 0 \text{ and } s_i \neq 0 \\ \Phi_S(i+1, i, k+1) & \text{if} j > 0, s_i \neq 0 \text{ and} s_i \neq s_j \\ \Phi_S(i+1, j, k) & \text{otherwise} \end{cases} \tag{C.7}$$

When $M(\epsilon) = 0$, it is an indication that there is no slope in the random walk. When $M(\epsilon) = 1$, it indicates that the random walk encountered maximal multimodality. Moreover, the expected number of optima can be calculated from the partial information content as $\lfloor \frac{nM(\epsilon)}{2} \rfloor$.

The information stability ($\epsilon*$) is the smallest value of ϵ where $H(\epsilon = 0)$. The higher the information stability, the higher largest possible difference between two neighboring points. Thus it provides an estimate of the magnitude of the optima on the fitness landscape encountered during the random walk.

The last measure, the density-basin information $h(\epsilon)$, can be calculated as

$$h(\epsilon) = -\sum_{p \in \{\overline{1}, 0, 1\}} P_{[pp]} log_3 P_{[pp]} \tag{C.8}$$

where pp is one of the sub-blocks $00, 11$, and $\overline{11}$. A high value of $h(\epsilon)$ indicates a high number of peaks occur within a small area of the landscape. A low value of $h(\epsilon)$ means that optima are isolated. Therefore, it gives an idea of the sizes of the basins of attraction of optima.

To summarise, higher values of information content, partial information content and information stability indicate higher degrees of epistasis and modality, defining a more rugged landscape which is harder to search. The density-basin information helps to understand the difficulty of the search space by characterizing the landscape at the regions near the optima.

Appendix D

The Strength Pareto Evolutionary Algorithm

In [Zit2001], Zitzler proposed the strength pareto evolutionary algorithm (SPEA2) as an extension of a previous well-known multi-objective evolutionary algorithm (SPEA) [Zit1999]. SPEA2 uses a regular population and an archive (external set). Starting with an initial population and an empty archive, it proceeds as follows. The fitnesses of all individuals are calculated. Then, all non-dominated indidividuals (defined below) are copied to the archive. If the archive is full, a truncation method is used to remove some individuals in the archive. On the contrary, if the archive is not filled up a procedure is used to add more individuals to the archive. Next individuals are selected from the archive genetic operators applied to them, after which the products are copied to the new population. The process is repeated until some terminating criteria are met. The algorithm for SPEA2 can be summarised as follows [Zit2001]

```
Input     $N$    (population size)
          $\overline{N}$    (archive size)
          $T$    (maximal number of generation)
Output    $A$    (nondominated set)

1)      Initialisation: Generate an initial population $P_0$ and
        create the archive (external set) $\overline{P}_0 =$. Set $t$=0.
2)      Fitness assignment: Calculate fitness values of individuals
        in $P_t$ and $\overline{P}_t$.
3)      Environment selection: copy all nondominated individuals
        in $P_t$ and $\overline{P}_t$ to $\overline{P}_{t+1}$. If the size of $\overline{P}_{t+1}$ exceeds $\overline{N}$ then
        Reduce $\overline{P}_{t+1}$ by means of the truncation operator,
        otherwise if the size of $\overline{P}_{t+1}$ is less than $\overline{N}$ then fill $\overline{P}_{t+1}$
        with dominated individuals in $\overline{P}_t$ and $\overline{P}_{t+1}$
4)      Termination: if $t \geq T$ or other stopping criteria
        are satisfied then set $A$ to the set of decision vectors
        represented by the nondominated individuals in $\overline{P}_{t+1}$. Stop.
5)      Mating selection: Perform binary tournament selection with
        replacement on $\overline{P}_{t+1}$ in order to fill the mating pool.
6)      Variation: Apply crossover and mutation operators to the
        mating pool and set $P_{t+1}$ to the resulting population.
        $t = t + 1$ and go to step 2
```

The fitness assignment is calculated as follows. Firstly, each individual i in the archive $\overline{P}_t$ and the population P_t is assigned a strength value $S(i)$ representing the number of solutions it dominates:

$$S(i) = |\{j / j \in P_t \cup \overline{P}_t \wedge i \succ j\}| \tag{D.1}$$

where $|\cdot|$ denotes the set cardinality and $\succ$ stands for the Pareto dominance relation. Next, the raw fitness $R(i)$ of each individual i is calculated as:

$$R(i) = \sum_{j \in P_t \cup \overline{P}_t, j \succ i} S(j) \tag{D.2}$$

Finally, the fitness $F(i)$ of each individual i is derived as

$$F(i) = R(i) + D(i) \tag{D.3}$$

where $D(i)$ is the density estimation defined by

$$D(i) = \frac{1}{\sigma_i^k + 2} \tag{D.4}$$

where σ_i^k is the distance between i and its k-th nearest neighbour point, usually, $k = \sqrt{N + \overline{N}}$. The density $D(i)$ is used to give encourage the individuals spread out through the Pareto frontier.

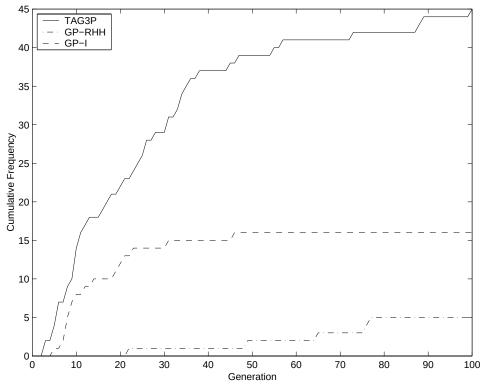
The environmental selection (step 3) is done as follows. The first step is to copy all nondominated individuals from the archive and population to the archive of the next generation. We note that a nondominated individual has fitness less than 1. Therefore, $\overline{P}_{t+1} = \{i/i \in P_t \cup \overline{P}_t \wedge F(i) < 1\}$. If the number of non-dominated individuals fits the size of the archive, the environmental selection is completed. Otherwise, there must be one of two situations: either the archive is too small or it is too large. In the first case, the next step is fill up the archive. It is done by sorting the multiset $P_t \cup \overline{P}_t$ according to fitness value, and copying the first $\overline{N} - |\overline{P}_{t+1}|$ individuals i with fitness $F(i) \leq 1$. In the second case, the next step is to truncate the archive. It is done by calling a procedure that iteratively removes individuals from $\overline{P}_{t+1}$ until its size returns to $\overline{N}$. In particular, at each iteration, the individual i is chosen for removal if $i \leq_d j$ for all $j \in \overline{P}_{t+1}$, where $i \leq_d j$ is defined as

$$i \leq_d j \ :\Leftrightarrow \begin{array}{l} \forall 0 < k < |\overline{P}_{t+1}| : \sigma_i^k = \sigma_j^k \vee \\ \exists 0 < k < |\overline{P}_{t+1}| \ : \ \left[\left(\forall 0 < l < k : \sigma_i^l = \sigma_j^l \right) \wedge \sigma_i^k < \sigma_j^k \right] \end{array} \tag{D.5}$$

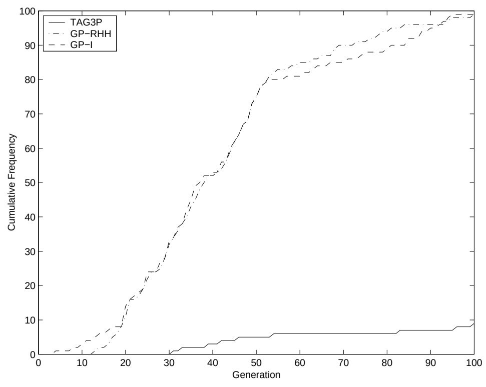
Appendix E

Some Supplementary Figures

In this chapter, some of the supplementary figures in chapter 10 are given.

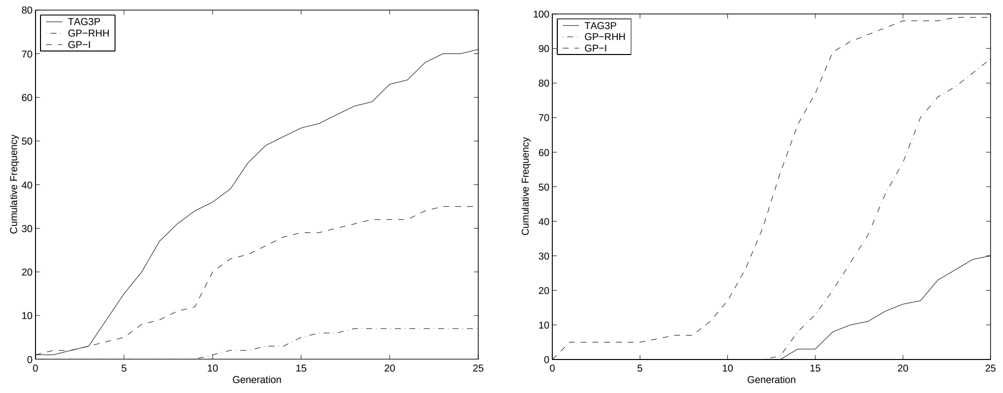


$SymbolicRegression$



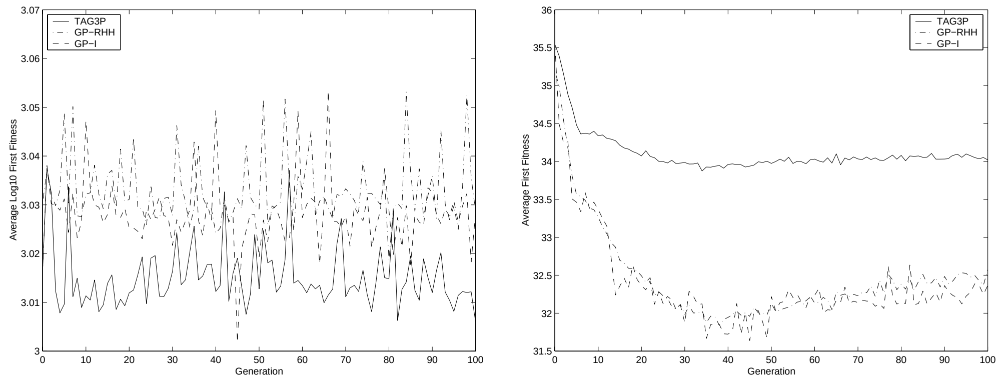
$6 - Multiplexer$

Figure E.1: Cumulative Frequencies (POPSIZE=250)



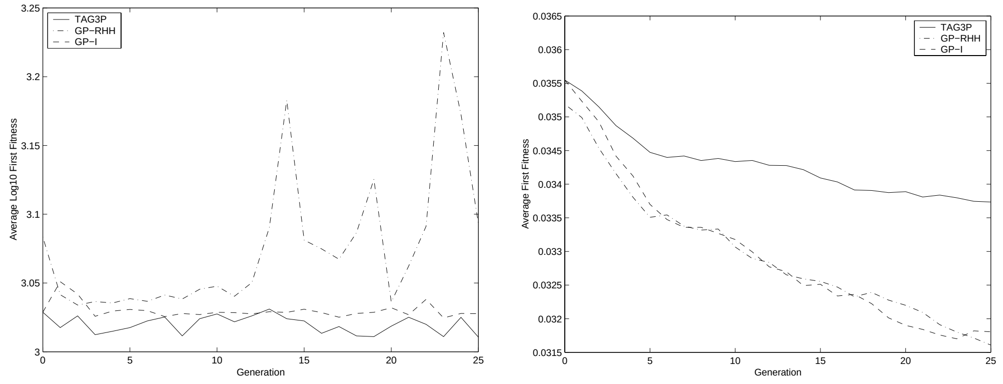
$SymbolicRegression$ $6 - Multiplexer$

Figure E.2: Cumulative Frequencies (POPSIZE=1000)



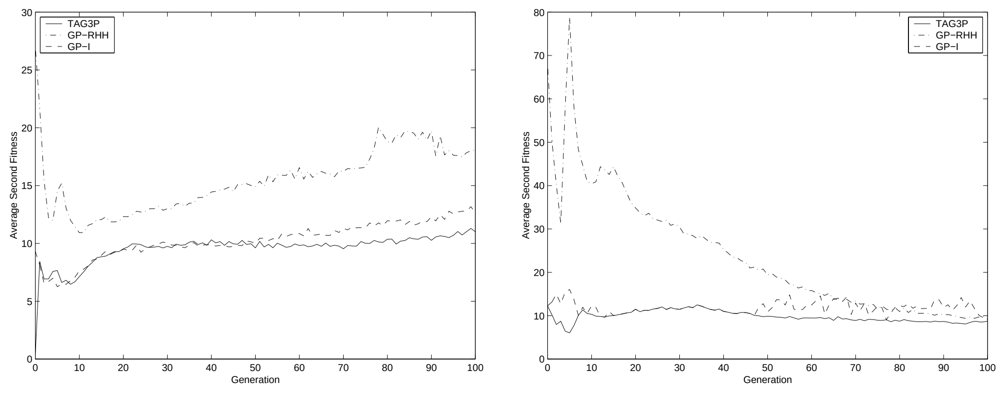
$SymbolicRegression$ $6 - Multiplexer$

Figure E.3: Average First Fitness (POPSIZE=250)



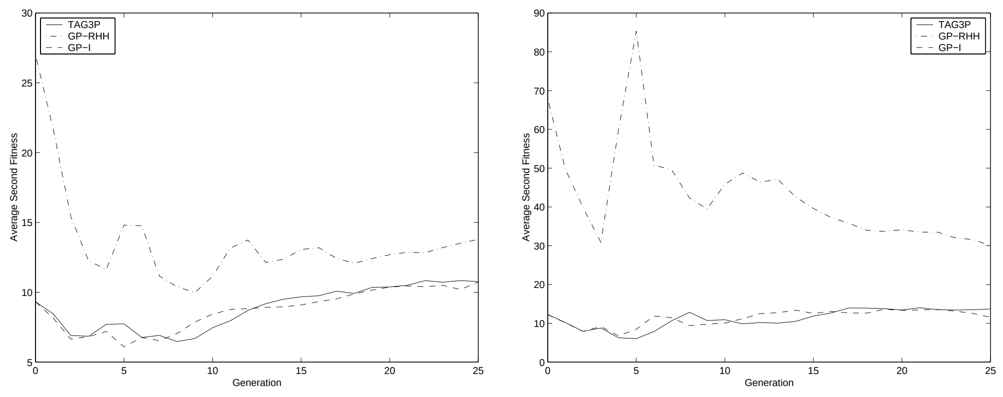
$SymbolicRegression$ $6 - Multiplexer$

Figure E.4: Average First Fitness (POPSIZE=1000)



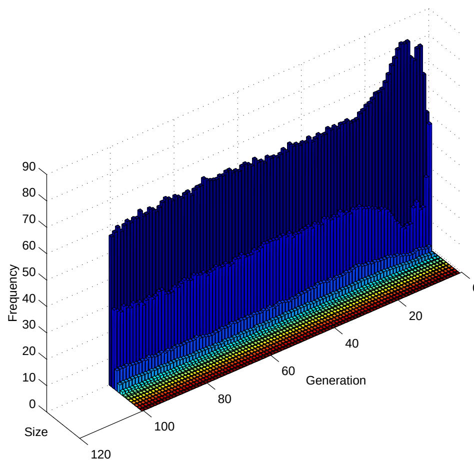
$SymbolicRegression$ $6 - Multiplexer$

Figure E.5: Average Second Fitness (POPSIZE=250)

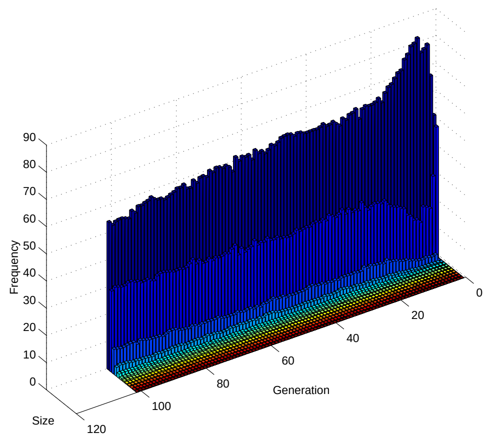


$SymbolicRegression$ $6 - Multiplexer$

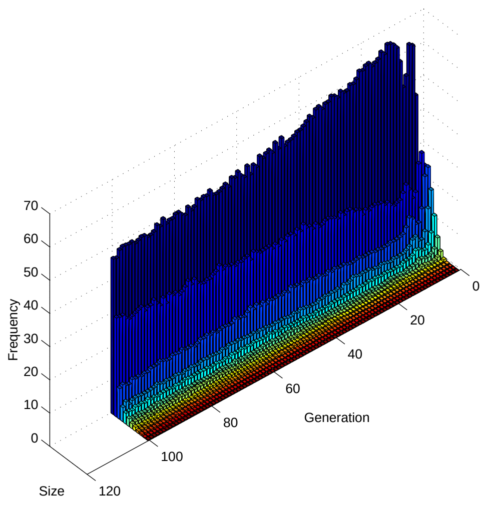
Figure E.6: Average Second Fitness (POPSIZE=1000)



$TAG3P$

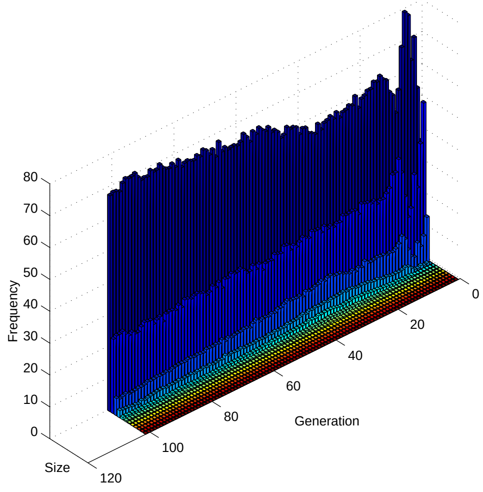


$GP-I$

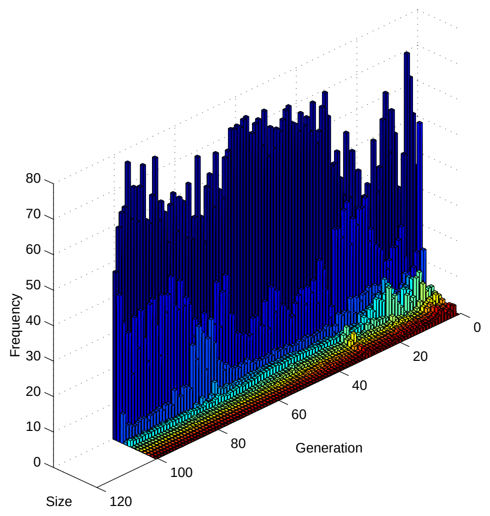


$GP-RHH$

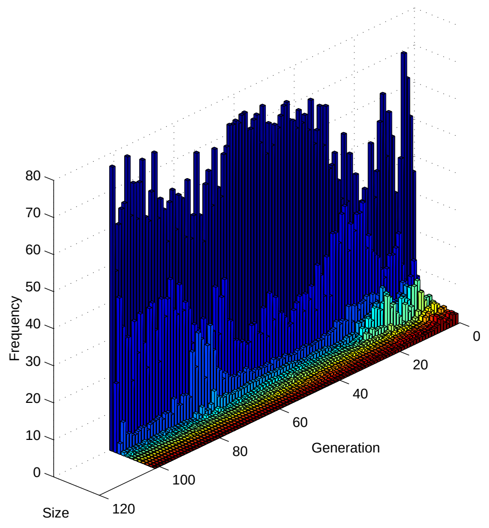
Figure E.7: Tree Size Frequencies for Symbolic Regression Problem, POP-SIZE=250



$TAG3P$

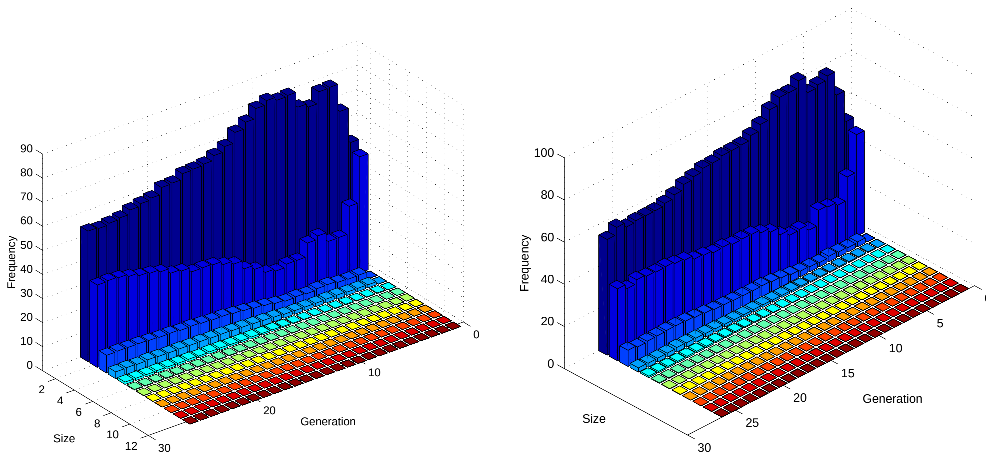


$GP - I$

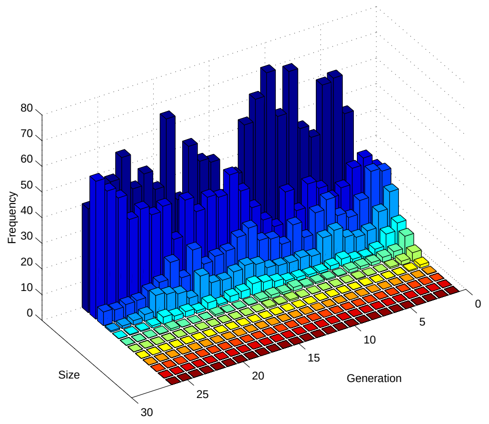


$GP - RHH$

Figure E.8: Tree Size Frequencies for 6-multiplexer Problem, POPSIZE=250



TAG3P *GP − I*



GP − RHH

Figure E.9: Tree Size Frequencies for Symbolic Regression Problem, POP-SIZE=1000

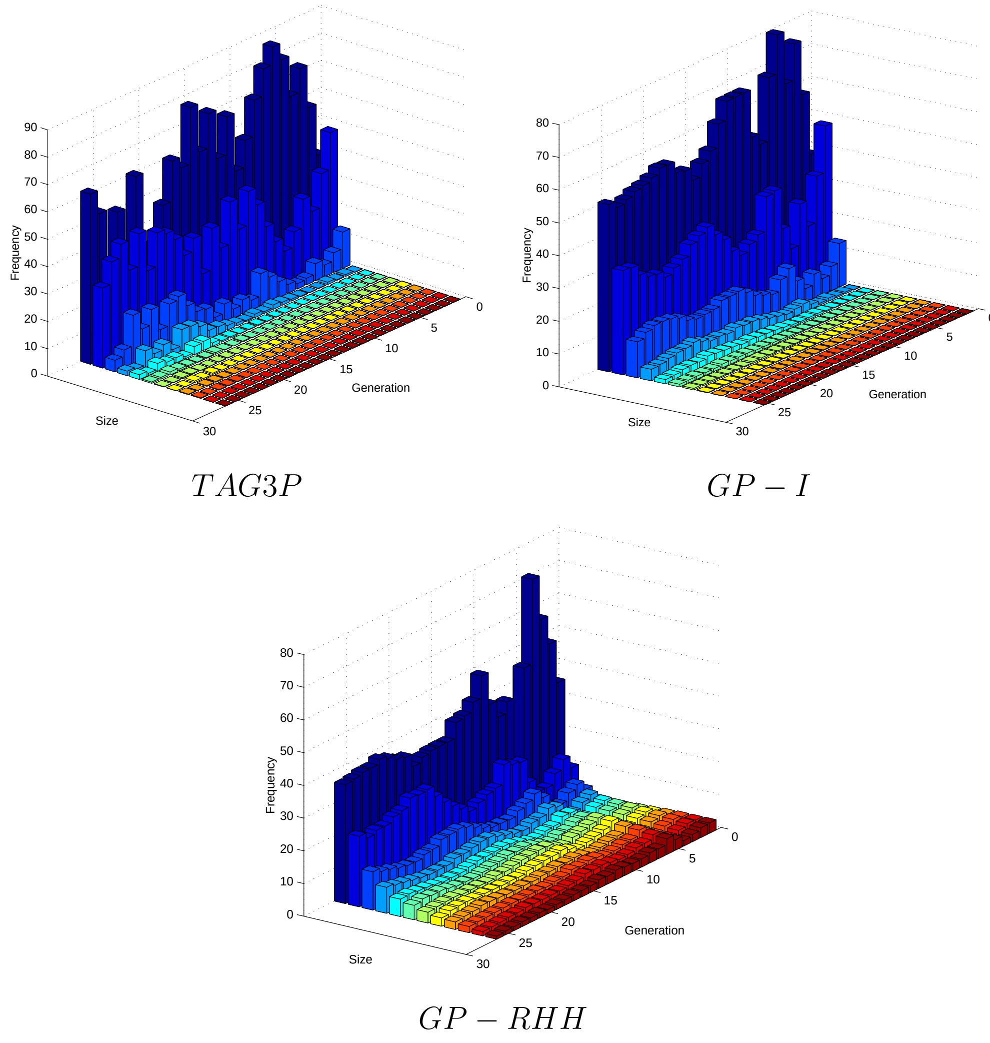


Figure E.10: Tree Size Frequencies for 6-multiplexer Problem, POPSIZE=1000